Universidad Autónoma de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Informática

TRABAJO DE FIN DE GRADO

APRENDIZAJE AUTOMÁTICO Y REDUCCIÓN DE LA DIMENSIONALIDAD EN PREDICCIÓN DE ENERGÍAS RENOVABLES

Autor: Iván de Andrés Tamé

Tutor: José Ramón Dorronsoro Ibero

Junio 2021

APRENDIZAJE AUTOMÁTICO Y REDUCCIÓN DE LA DIMENSIONALIDAD EN PREDICCIÓN DE ENERGÍAS RENOVABLES

Autor: Iván de Andrés Tamé
Tutor: José Ramón Dorronsoro Ibero

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Junio 2021

Iván de Andrés Tamé
Aprendizaje automático y reducción de la dimensionalidad en predicción de energías renovables

Iván de Andrés Tamé
IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos.

El tiempo es oro.

Benjamin Franklin

Agradecimientos

Este trabajo no hubiese sido posible sin la colaboración de la Cátedra UAM-IIC en ciencia de datos y aprendizaje automático, que me ha dado la oportunidad trabajar con sus recursos. Quiero agradecer especialmente a José por la guía que me ha dado, tanto en este trabajo como durante la carrera.

También quiero agradecérselo a mi familia por apoyarme cada día y ser comprensivos conmigo durante estos años de carrera, especialmente a mis padres y mi abuela.

Por último, pero no por ello menos importante a todos los grandes amigos que he hecho durante la carrera, siendo ellos de lo mejor que me llevo de esta. Especialmente a Julio, Pablo, Íñigo y Alejandro por haber sido todos un apoyo de todos en cualquier situación.

Resumen

Resumen

Con el incremento de la población y el consumo de la misma, las formas de obtener energía han ido evolucionando desde hace años, y las energías renovables se van abriendo camino, puesto que suponen un factor diferenciador en el camino hacia un futuro energéticamente sostenible. Aun así el uso de estas energías está lejos de ser 100 % aprovechable, debido a factores como por ejemplo, la dificultad a la hora de predecir componentes que los seres humanos no podemos controlar.

Gracias a la potencia de los ordenadores actuales, y a las técnicas de aprendizaje automático, la predicción de estos componentes ajenos a nuestro control, es cada vez más efectiva. Aun con esto, la cantidad de datos que se necesitan para aplicar estas técnicas y tener un buen resultado puede llegar a ser enorme, dificultando tanto su procesamiento como su envío. Para tratar de solucionar este problema, este trabajo tratará de aplicar técnicas de reducción de dimensionalidad, y comprobar la eficiencia de las predicciones tras esto.

Para este trabajo se han utilizado datos numéricos reales sobre predicciones meteorológicas, así como diferentes modelos de regresión. Estos procesos se han realizado enteramente usando el lenguaje de programación *Python*, haciendo especial uso de la librería **scikit-learn** para los modelos de aprendizaje automático.

Palabras Clave

Reducción de dimensionalidad, Aprendizaje automático, Predicción, Regresión, Meteorología, Python, Sklearn

Summary

With the increase in population and its consumption, the ways of obtaining energy have been evolving for years, and renewable energies (clean energies) are making headway, since they represent a differentiating factor on the way to an energetically sustainable future. Even so, the use of these energies is far from being 100 % usable, due to factors such as, for example, the troubles in predicting components that humans cannot control.

Thanks to the power of today's computers and machine learning techniques, the prediction of these components beyond our control is becoming increasingly effective. Even so, the amount of data needed to apply these techniques and get a good result can be enormous, making it difficult to process and send. To try to solve this problem, this work will try to apply dimensionality reduction techniques, and check the efficiency of the predictions after that.

For this work, real numerical data on weather forecasts have been used, as well as different regression models. These processes have been performed entirely using the *Python* programming language, making special use of the **scikit-learn** library for machine learning models.

Key Words

Dimensionality Reduction, Machine Learning, Prediction, Regression, Meteorology, Python, Sklearn

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

1.1. Motivación del proyecto

El uso de energías renovables es cada vez más rentable, y la mayoría de países quieren invertir en las mismas, puesto que son una fuente limpia e inagotable de conseguir energía, en España, a día de hoy las energías renovables superan el 40 % de la producción nacional. Por esto mismo las compañías eléctricas, cada vez están más interesadas en optimizar la cantidad de energía que pueden producir. El problema se encuentra en que este tipo de energías son muy irregulares, y dependiendo de la localización en la que se instalen las infraestructuras encargadas de producirlas (por ejemplo un parque eólico), el rendimiento y cantidad de energía obtenida puede variar enormemente.

Para saber donde situar estas infraestructuras, se ha de tener en cuenta que las condiciones sean favorables, y gracias a las tecnologías actuales y a los esfuerzos de muchas instituciones, actualmente se pueden medir estas condiciones en diferentes puntos geográficos, transformar en valores numéricos y guardarlos en conjuntos de datos. El problema con el que nos encontramos es que estos datos pueden ser muy abundantes, y saber que valores tener en cuenta sea una tarea difícil, y en caso de usar todos, se puede convertir en un proceso computacionalmente costoso.

Teniendo esto en cuenta, el objetivo de este trabajo ha sido tanto el de observar como se adapta el proceso de reducción de dimensionalidad de un conjunto de datos a este problema, así como probar diferentes herramientas de predicción de los datos.

Este trabajo se ha realizado con la colaboración de la Cátedra UAM-IIC (Instituto de Ingeniería del Conocimiento), utilizándose datos del parque eólico de Sotavento proporcionados por ellos.

1.2. Objetivos y enfoque

Como se ha explicado antes el objetivo de este trabajo es el de aplicar diferentes metodologías de aprendizaje automático y *machine learning* sobre el problema de la predicción de energías renovables, en este caso centrándonos en la energía eólica. Estas metodologías se aplicarán sobre los datos reducidos y sin reducir para comprobar si el proceso de reducción de dimensionalidad puede aportar una ventaja, ya sea en calidad de la predicción o en rendimiento de la misma.

Para esto hemos realizado transformaciones a los datos proporcionados, para reducir su dimensionalidad, y luego hemos probado diferentes herramientas de predicción usando el lenguaje de programación *Python*.

1.3. Organización de la memoria

Este trabajo de fin de grado, además de esta primera parte de introducción, se estructurará de la siguiente manera:

Capítulo 1: Motivación y objetivos del proyecto. En este capítulo se dará una idea general de lo que tratará el presente trabajo.

Capítulo 2: Conceptos básicos sobre energías renovables y predicción meteorológica. En este capítulo se hablarán de varios conceptos relacionados con las energías renovables, que a pesar de no todos ser imprescindibles para el objetivo del trabajo, es interesante tenerlos en cuenta.

Capítulo 3: Introducción al aprendizaje automático y reducción de dimensionalidad. En este capítulo se detallarán los conceptos matemáticos y de *machine learning* en los que se fundamentará el trabajo.

Capítulo 4: Descripción de los experimentos elaborados, así como las herramientas utilizadas. En este capítulo se mostrarán tanto el proceso como el resultado de los experimentos realizados.

Capítulo 5: Conclusiones generales y posible trabajo futuro.

2

Predicciones de meteorología y de energía renovable

2.1. Introducción a la energía renovable

Las energías renovables se definen como aquellas que proceden de fuentes no fósiles, siendo las más relevantes y conocidas la energía solar y eólica. Las ventajas que aportan estas con respecto las no renovables, son bastante considerables, pero destacaremos las 3 siguientes:

- Su aprovechamiento afecta de manera muy sutil al medio ambiente, dado que no generan residuos.
- Son fuentes de energía ilimitada, por ello también se las conoce como fuentes de energía inagotable, ya que no se agotan con su consumo.
- Se trata de fuentes de energía que se encuentran en la mayor parte del mundo, por lo que no genera dependencia del exterior, como por ejemplo el petróleo, el cual es más abundante en ciertos países que tienen el monopolio del mismo.

Por otro lado, el desarrollo de energías renovables es imprescindible para ayudar a combatir el cambio climático y sus efectos. El uso de energías fósiles aumenta de manera exponencial la cantidad de CO_2 que vamos enviando a la atmósfera, haciendo que el ritmo sea insostenible. Este gran consumo choca bastante con el hecho de que no toda la población tiene acceso a electricidad, puesto que transportarla es caro, y no llega a todas partes, problema que con las energías renovables también se combatiría.

Teniendo en cuenta el crecimiento en la necesidad de energía, acompañado con el hecho de que las energías no renovables, como su propio nombre indica, se van consumiendo sin reponerse, el estudio de las nuevas energías naturales debería ser una prioridad para asegurar así un futuro sostenible.

2.1.1. Energía eólica

La energía eólica es una de las energías renovables más desarrolladas. Genera electricidad a través de la fuerza del viento, mediante la utilización de la energía cinética producida por efecto

de las corrientes de aire. Se trata de una fuente de energía limpia e inagotable, que reduce la emisión de gases de efecto invernadero y preserva el medioambiente.

Desde principios del siglo XX se produce energía a través de los aerogeneradores, haciendo que el viento mueva una hélice y, mediante un sistema mecánico, se haga girar el rotor de un generador que produce energía eléctrica. Los aerogeneradores suelen agruparse en concentraciones denominadas parques eólicos con el fin de lograr un mejor aprovechamiento de la energía, lo que reduce su impacto ambiental. [8]

La energía eólica, aunque sorprenda, se puede considerar una variante de la energía solar, puesto que deriva calentamiento diferencial de la atmósfera así como de las irregularidades de relieve de la superficie terrestre. De toda la energía solar que recibe la tierra solo una pequeña parte se transforma en energía eólica, y a pesar de esto supera con creces las necesidades actuales de electricidad. La energía eólica es un recurso muy variable, tanto en el tiempo como en el lugar, pudiendo cambiar mucho en distancias muy reducidas. En las zonas costeras y las cumbres de las montañas generalmente es donde más se aprovecha esta energía.

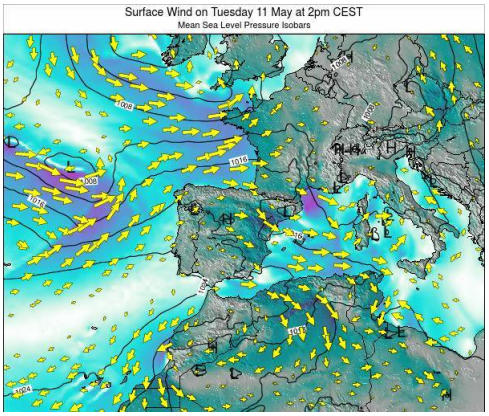

Figura 2.1: Dirección del viento cerca de la península ibérica en la superficie (Imagen extraída de surf-forecast [1])

2.2. Predicción numérica del tiempo

La predicción numérica del tiempo utiliza modelos físico-matemáticos para predecir el estado futuro de la atmósfera, basándose en el estado actual y pasado de la misma. Estas técnicas se volvieron una realidad tangible con la llegada de la computación, puesto que la cantidad de cálculos que se necesitan son muy grandes. Estos cálculos se realizan basándose en que la atmósfera es un fluido, y por lo tanto se pueden crear simulaciones que usen la mecánica de fluidos y la termodinámica.

Una simulación consiste en reproducir el comportamiento de un sistema a partir de un modelo. Dado que en la computación nos interesa saber los valores cuantitativos del sistema, aplicaremos modelos matemáticos, que utilizan ecuaciones diferenciales para representarlo. En la práctica si de estas ecuaciones se obtienen resultados exactos solo puede implicar que el sistema era demasiado simple, o que las ecuaciones lo representan únicamente de forma simplificada.

Un concepto importante para que a partir de las ecuaciones se llegue a un resultado adecuado, es la inicialización del estado. Este estado inicial en la meteorología es muy complejo de definir por todos los datos que se manejan. Estos se obtienen desde diferentes estaciones meteorológicas situadas en todo el globo, y miden diferentes valores físicos del ambiente, como el viento, temperatura, humedad, etc. También es muy recurrente el uso de aviones y globos sonda para este proceso, gracias a que pueden proporcionar estos datos con diferentes medidas.

La cantidad de datos que se recibe diariamente de las diferentes estaciones es tremendamente alta e irregular, así que han de ser interpoladas en forma de malla, para que en estas mallas se puedan discretizar las ecuaciones diferenciales y poder así resolverlas. Estas rejillas serán más completas a más puntos recojan y más distancia cubran; a esto se le denomina tener una buena *resolución de modelo*. A mayor resolución, mejores serán los resultados, pero también habrá que hacer una mayor cantidad de operaciones.

2.2.1. European Centre for Medium-Range Weather Forecast

El *European Centre for Medium-Range Weather Forecast (ECMWF)* es una organización independiente e intergubernamental, la cual tiene como objetivo principal ser capaz de generar una previsión meteorológica fiable a medio y largo plazo [9].

El centro se creó en el año 1975, por 18 países de origen europeos en su mayoría, a los que luego se añadieron otros 4, haciendo un total de 22 países miembros en la actualidad, a los que hay que sumar otros 12, que a pesar de no ser miembros oficiales, colaboran con el ECMWF.

La principal misión del ECMWF consiste en los 3 siguientes puntos:

- Producir predicciones numéricas del clima y monitorizar el sistema terrestre.
- Investigación técnica y científica para mejorar las predicciones.
- Mantener un archivo histórico de datos meteorológicos.

Para esto el centro produce análisis y predicciones basadas en conjuntos, así como su probabilidad de que ocurran. Estas predicciones cubren espacios de tiempo desde diarios hasta anuales. La gran cantidad de datos que manejan ha hecho que comiencen a dar la oportunidad de analizar y predecir otros aspectos del ambiente natural además de la predicción meteorológica, como la composición atmosférica, el cambio climático y predicción de peligro ante inundaciones o incendios. Para muchas de estas aplicaciones el ECMWF colabora con el proyecto *Copernicus*, el principal programa europeo de monitorización de la Tierra.

El sistema de previsión del ECMWF consta de tres componentes: un modelo de circulación general (acoplado a un modelo de olas oceánicas), un sistema de asimilación de datos y, desde 1992, un sistema de previsión de conjuntos. El primer modelo numérico del ECMWF era un modelo de puntos de rejilla y la resolución correspondía a un área de la rejilla de unos 200 km^2. Este modelo de puntos de rejilla fue sustituido por un modelo denominado *modelo espectral*. La técnica espectral era más precisa que el modelo de puntos de rejilla con el mismo coste computacional. Con las altas resoluciones actuales, tanto con los modelos de puntos de rejilla como con los espectrales, ya no hay diferencias significativas en la precisión.

El modelo que sigue actualmente el ECMWF, consiste en tres componentes, una dinámica, otra física y una basada en las olas oceánicas. La formulación del modelo puede resumirse en seis ecuaciones físicas básicas:

- La **ley de los gases**, que indica la relación entre la presión, la densidad y la temperatura.
- La **ecuación hidrostática**, que muestra la relación entre la densidad del aire y el cambio de presión con la altura.
- La **ecuación de continuidad**, que expresa la conservación de la masa y determina la velocidad vertical y el cambio en la presión superficial.

- La **ecuación de movimiento**, que describe cómo cambia el momento de un paquete de aire debido al gradiente de presión y a la fuerza de *Coriolis*.
- La **ecuación termodinámica**, que expresa cómo se produce un cambio en la temperatura de una parcela de aire por enfriamiento o calentamiento adiabático (sistema el cual no intercambia calor con su entorno), debido a desplazamientos verticales.
- La **ecuación de conservación de la humedad**, que supone que el contenido de humedad de una parcela de aire es constante, salvo las pérdidas por precipitación y condensación o las ganancias por evaporación de las nubes y la lluvia o de los océanos y continentes.

De estas seis ecuaciones, las dos primeras son de diagnóstico y nos hablan de la relación estática entre diferentes parámetros, mientras que las cuatro siguientes son pronósticos y describen los cambios con el tiempo de las componentes horizontales del viento, la temperatura y el contenido de vapor de agua de una parcela de aire, y de la presión de superficie. [10]

2.3. Formatos de datos meteorológicos

En esta sección se hablará de los formatos más típicos en los que se suele almacenar los datos meteorológicos, así como los usados en este trabajo. En todos los procesos de *machine learning*, el tratamiento previo de los datos es una de las partes más importantes, y el formato en el que vengan dados estos puede hacer que el procesamiento previo sea más o menos costoso.

Formato GRIB

El formato **GRIB** (GRIdded Binary, o General Regularly-distributed Information in Binary form), es un formato usado frecuentemente en la meteorología para archivar y predecir información sobre el tiempo. Fue creado por la Organización Meteorológica Mundial (con siglas WMO en inglés) en 1985.

Estos ficheros GRIB, están separados en mensajes o regiones, que a su vez se separan en secciones, las cuales pueden dar información y/o datos reales [11]. Estas secciones son:

0. Sección indicadora. Identifica el inicio del mensaje en el fichero GRIB, así como la longitud total del mismo y el número de edición de GRIB.
1. Sección de definición del producto. Da información general sobre el mensaje, como el área que cubren los datos, la variable que se tiene en cuenta, información sobre la fecha y hora, etc. Generalmente se suele separar en 28 octetos, pero puede ser mayor.
2. Sección de descripción de la matriz. Ofrece información sobre la matriz de datos. Es una sección opcional, pero muy recomendada.
3. Sección del mapa de bits. Incluye un mapa de bits, o una referencia a uno, definido previamente por el centro de datos donde se creó el archivo. Consiste por lo tanto, en una cadena de bits, que hace referencia a la matriz de datos, donde si hay un *1*, implica que esa posición en la matriz contiene datos, y si hay un *0* que no. Esta sección también es opcional.
4. Sección de datos binarios. Aquí se encuentran los datos comprimidos e información sobre cómo reconstruirlos.

5. Sección final. En esta sección se encuentra el código ASCI *7777* para indicar el final del mensaje. No es recomendable buscar esta cadena en el archivo para encontrar el final, puesto que la cadena de bits se puede encontrar en cualquier sitio del archivo, dando lugar a fallo.

A la información que se encuentra en estos mensajes se puede acceder mediante el uso de librerías de lenguajes de programación así como la herramienta *GRIB_ API*.

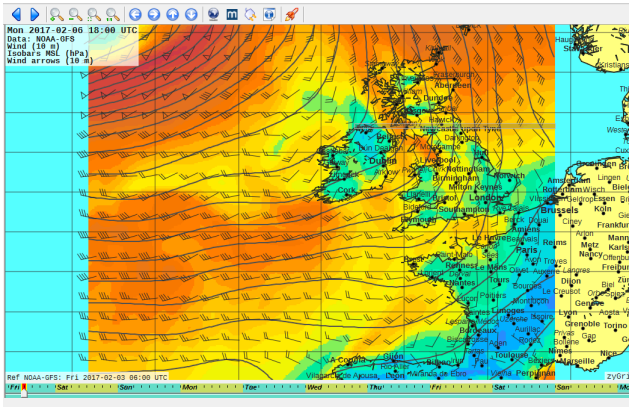

Figura 2.2: Mapa dibujado a partir de datos en formato GRIB usando la herramienta ZyGrib (Imagen extraída de weather mailsail [2])

Formato NetCDF

NetCDF es un conjunto de bibliotecas de software y formatos de datos autodescriptivos que permiten crear, acceder y compartir datos científicos multidimensionales en formato de matrices. NetCDF fue desarrollado por *Unidata*, que proporciona datos y herramientas de software para su uso en la educación e investigación en geo-ciencias. Unidata forma parte de los *UCAR Community Programs (UCP)* llevados a cabo por la *Corporación Universitaria para la Investigación Atmosférica (UCAR)*. Unidata está financiada principalmente por la *National Science Foundation* [12]. Un archivo NetCDF tiene las siguientes características:

- Autodescriptivo. Un archivo netCDF incluye información sobre los datos que contiene.
- Portátil. Se puede acceder a un archivo netCDF desde ordenadores con diferentes arquitecturas.
- Escalable. Se puede acceder a pequeños subconjuntos de grandes *datasets* de forma eficiente a través de las interfaces de netCDF, incluso desde servidores remotos.
- Concatenable. Los datos pueden añadirse a un archivo netCDF correctamente estructurado sin necesidad de copiar el conjunto de datos o redefinir su estructura.
- Compartible. Un escritor y varios lectores pueden acceder simultáneamente al mismo archivo netCDF.
- Compatible. El acceso a todas las formas anteriores de datos netCDF será compatible con las versiones actuales y futuras del software.

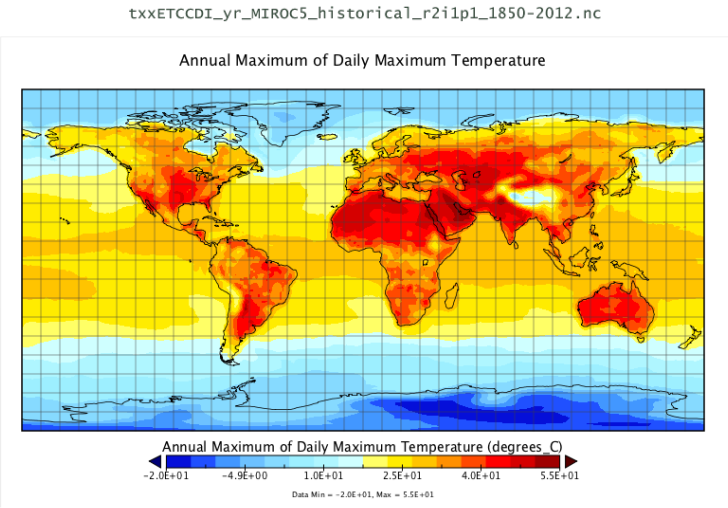

Figura 2.3: Mapa dibujado a partir de datos en formato netCDF (Imagen extraída de Osgeo [3])

Formatos alternativos y DataFrames

Mientras que los formatos explicados previamente ofrecen información adicional a los datos explícitos, se tratan de formatos complejos, que en casos como en este TFG aportan información que no será utilizada y por lo tanto complican el proceso de extracción de datos. Para simplificar las cosas, los *datasets* muchas veces se encuentran en formatos muy simples, los cuales a lo mejor carecen de características importantes, como ser autoexplicados, pero a cambio son muy simples de almacenar. Uno de los formatos más típicos, y en el cual encontraremos muchos *datasets*, una vez han sido tratados inicialmente, es el formato *Coma Separated Values (CSV)*. El formato *CSV*, como ya se ha mencionado, es muy sencillo, y no transforma los datos a otro sistema de representación, ni indica la posición de los bytes. Por ser tan simple, también puede presentar problemas como por ejemplo cuando se quiere guardar una cadena de caracteres que contiene comas. Dependiendo de la implementación usada de *CSV*, estos estarán solucionados o no.

Para el tratamiento de los datos, inicialmente es buena idea representarlos de alguna manera con la que luego sea simple trabajar con ellos. Para esto la librería **pandas** del lenguaje de programación *Python*, nos proporciona el objeto *DataFrame*. Este objeto nos permite agrupar los datos en tablas bidimensionales e indexadas, lo que en nuestro caso será realmente útil para ordenar los datos de los diferentes campos eólicos por hora y fecha. Dado que los datos vienen por lo general en diferentes tipos de ficheros es necesario usar funciones específicas que nos lo puedan transformar en un objeto *pandas* directamente, como por ejemplo *read_ csv* en el caso de que los datos estén en formato *csv* (uno de los formatos más típico).

En ciertos casos, el *dataset* con las *features* y el de los *targets* vienen por separado. En estos casos es una buena práctica el juntar ambos en un mismo *dataset*, aunque luego se vaya a hacer uso de ellos por separado. Esto es así porque se puede dar el caso de que no vengan ordenados de la misma manera, o que uno de los *dataset* tenga inconsistencia en alguna de sus filas (es decir, que alguno de los ejemplos no se haya registrado) y el otro no lo tenga. Este proceso nos ayuda a prevenir errores posteriores. Para realizar esto **pandas** ofrece la función *concat*, la cual concatena objetos **pandas** a lo largo de un eje particular (en nuestro caso por columnas), similar a un *join* de conjuntos.

3

Aprendizaje automático y reducción de dimensionalidad

3.1. Modelos de aprendizaje automático

En las ciencias de la computación se denomina aprendizaje automático (del inglés *machine learning*) a la rama derivada de la inteligencia artificial enfocada en diseñar algoritmos que sean capaces de aprender por sí mismos. En este contexto, entendemos aprender cuando el resultado mostrado por el algoritmo mejora con la experiencia, es decir cuando a partir de ejemplos previos, el mismo algoritmo es capaz de dar un resultado cada vez más acertado. Por lo tanto, el objetivo final de los investigadores que trabajan en este campo es el de aplicar métodos matemáticos para automatizar partes del método científico.

Podemos distinguir diferentes tipos de algoritmos según como estén categorizados los datos que se les proporcionen. Los más conocidos e importantes para el proyecto son los siguientes:

Aprendizaje supervisado

Este tipo de aprendizaje se realiza con los datos y su correspondiente salida para validación. Con estos datos se crean dos grupos, uno de entrenamiento (*training dataset*), del cual sacaremos los datos para que el algoritmo cree una función que relacione de la forma más precisa posible las entradas con las salidas esperadas. Y otro de validación (*test dataset*), con el que se comprobará lo eficiente que es la función creada anteriormente.

Dentro de este tipo de aprendizaje podríamos agrupar los problemas a resolver como dos:

- **Problemas de regresión**: Problemas en los cuales la salida está formada por una o más variables con valores continuos. Por ejemplo, predecir los puntos que va a hacer un equipo de baloncesto con respecto sus anteriores partidos.
- **Problemas de clasificación**: Problemas en los cuales la salida consiste en asignar el dato a uno de los grupos establecidos al inicio del problema, es decir, predecir una variable discreta. Por ejemplo predecir si ese mismo equipo de baloncesto, va a ganar o perder el siguiente partido.

Para construir un modelo de regresión por lo general tendremos una muestra $S = \{x^p, y^p\}, 1 \leq p \leq N$ con las características (**features**) $x^p \in R^d$, con objetivo (**target**) y^p. Con esto querremos construir un modelo $\hat{y} = f(x)$, tal que $\hat{y}^p = f(x^p) \approx y^p$, que significa que queremos hacer una regresión de la y con respecto la x. La opción más típica es la de seleccionar una f que minimice el error medio al cuadrado (**MSE**) de la muestra.

$$\hat{e}(f) = \hat{e}_S(f) = \frac{1}{2N}\sum_{p=1}^{N}(y^p - f(x^p))^2$$

Aprendizaje no supervisado

En este tipo de aprendizaje los datos vienen dados sin su correspondiente salida, y por lo tanto no se tiene información sobre lo correcta que es la salida de la función aplicada a los datos. Este aprendizaje se utiliza por lo tanto para la búsqueda de patrones entre los datos dados.

Aprendizaje semi-supervisado

Consiste en una combinación de los dos tipo de aprendizaje anteriores, en el que se tiene en cuenta los datos categorizados y sin categorizar. Esto se hace así porque incluir una pequeña cantidad de datos categorizados, junto a un conjunto de varios sin categorizar, puede aumentar la exactitud del aprendizaje. A su vez, etiquetar imágenes es costoso, así que a veces es más rentable solo realizar este proceso con algunas.

3.1.1. Regresión lineal y regresión Ridge

En estadística un modelo lineal es aquel que se usa para aproximar la relación de dependencia lineal entre una variable dependiente $\hat{y}$, las variables independientes x_i y un término aleatorio ε. Para cada variable, este modelo puede ser expresado como:

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n + \varepsilon$$

donde $\hat{y}$ es el resultado del modelo, x_i son las variables independientes, explicativas o regresores introducidas al modelo, w_i los parámetros que miden cuánto influye las x_i sobre el resultado, w_0 es un término constante y ϵ representa todos los factores distintos de x que pueden influir sobre la variable dependiente, pero los cuales no son relevantes individualmente.

En muchos casos, como por ejemplo para obtener los valores de w, nos será útil ver esta fórmula de manera matricial, de la forma

$$\hat{Y} = Xw + \epsilon$$

siendo $\hat{Y}$ el vector de resultados del modelo y X el vector con todas las variables independientes (x_i), siendo en todo caso $X_0 = 1$.

En el caso de tener únicamente dos variables, la forma de visualizar esta representación sería trazar la recta que minimice el *error cuadrático medio (ECM)* de cada uno de los conjuntos. La función resultante en ese caso tendría la siguiente forma:

$$\hat{y} = w_0 + w \cdot x$$

donde w_0 es el desplazamiento sobre el origen en una dimensión, y w la pendiente de la recta. Si centramos las variables de manera que $E[x_i] = E[y] = 0$, entonces $w_0 = 0$. Quedamos por lo tanto con un modelo más simplificado: $\hat{y} = w \cdot x$. En la práctica siempre normalizaremos x, para tener media = 0 y desviación típica = 1.

Estando X e Y centrados, siendo Y un vector tamaño $1 \times N$ y organizamos la muestra S en una matriz de datos $N \times d$, el error cuadrático medio se obtiene mediante:

$$\hat{e}(w) = \frac{1}{2N}\sum_{p=1}^{N}(w \cdot x^p - y^p)^2 = \frac{1}{2N}(Xw - Y)^t(Xw - Y) = \frac{1}{2N}(w^t X^t X w - 2w^t X^t Y + Y^t Y)$$

Ahora para minimizarlo, queda calcular $\nabla\hat{e}(w) = 0$, con lo que resolviendo se puede comprobar que,

$$\nabla\hat{e}(w) = \frac{1}{N}X^t X w - \frac{1}{N}X^t Y = \hat{R}w - \hat{b},$$

donde $\hat{R}$ es la matriz de covarianza de la muestra $y\hat{b} = X^t Y$. Para buscar un $\hat{w}$ óptimo ($\hat{w}^*$), tendremos que resolver la ecuación $\hat{R}w - \hat{b} = 0$, que siendo $\hat{R}$ invertible, se calcula de la siguiente manera:

$$\hat{w}^* = \hat{R}^{-1}\hat{b} = (X^t X)^{-1}X^t Y$$

La principal dificultad que hay detrás de aplicar una regresión lineal en el aprendizaje automático, es la de encontrar el valor de todos los w_i, puesto que este método no vale el 100 % de las veces. Para obtenerlo tenemos varios métodos:

Mínimos cuadrados ordinarios

El método de mínimos cuadrados ordinarios (*OLS* por sus siglas en inglés), se basa en observar la diferencia entre el valor observado y el valor obtenido en el modelo (lo que hasta ahora hemos estado denominando ϵ, conocido también como residuo). La suma de los cuadrados de los residuos (siglas *RSS* en inglés), viene dada por la siguiente fórmula:

$$RSS = e_R(w) = \frac{1}{2N}\sum_{p=1}^{N}(y^p - x^p \cdot w)^2$$

Si tratamos X de manera matricial, como hemos indicado anteriormente, y se trata de una matriz $N \times k$, en la que cada una de las filas es un vector de datos correspondientes al entrenamiento, e Y es un único vector tamaño N con sus salidas. La fórmula quedaría entonces:

$$RSS = e_R(w) = (Y - Xw)^t(Y - Xw)$$

Si se da el caso de que la matriz $X^T X$ se puede invertir, entonces al minimizar RSS se obtiene que los coeficientes tienen una solución única, representada por la fórmula

$$w = (X^T X)^{-1}X^t Y$$

Regresión Ridge

La *Ridge Regression* es una técnica para analizar múltiples datos de regresiones que sufren de multicolinealidad. La misma penaliza el tamaño de las w_i, y tata de minimizar el tamaño de la RSS al añadir el cuadrado de la media de los coeficientes. En el caso de que la matriz $\hat{R}$ no sea invertible usaremos el método de la *regresión Ridge*. Para solucionar esto podemos trabajar con $\hat{w}^* = (X^t X + \lambda I)^{-1} X^t Y$ para algunos valores de λ. La función a minimizar es:

$$RSS = e_R(w) = \frac{1}{2N} \sum_{p=1}^{N} (y^p - x^p \cdot w)^2 + \frac{\lambda}{2} ||w||^2$$

donde λ es un parámetro para controlar la cantidad de penalización. Cuando mayor es este, menores serán los coeficientes. También se puede observar que cuando $\lambda = 0$, y por lo tanto tiene una penalización nula, el resultado es el mismo que por mínimos cuadrados ordinarios (OLS).

Por lo tanto, este método es útil cuando sospechemos que varios atributos de la entrada pueden estar relacionados entre ellos puesto que, al minimizar, los coeficientes la correlación entre ellos será menos notable, y por lo tanto el modelo generalizará más y evitará el overfitting.

3.1.2. Mínimos cuadrados parciales

Los mínimos cuadrados parciales o *PLS regression* por sus siglas en inglés (*Partial least squares regression*), es un método de regresión basado en la covarianza. Es especialmente útil en casos donde el número de variables explicativas es alta. Se entiende como variable explicativa, un tipo de variable independiente, la cual no es claro que no dependa de ninguna otra variable, es decir que tenga una alta correlación.

La idea principal de la *PLS regression* es crear un *dataset* con Z componentes a partir de una tabla con N variables, siendo siempre $Z < N$, aplicando los algoritmos *PLS1* y *PLS2*. *PLS1* corresponde al caso en el que únicamente se tiene en cuenta una variable dependiente. *PLS2* en contraposición, corresponde a cuando hay varias de estas variables. Se puede codificar de manera que *PLS1* sea un sub-problema de *PLS2*.

El algoritmo *PLS1* consiste en extraer la primera componente, y a partir de ella el resto, de manera que no estén correlacionadas. La primera componente se describe como:

$$Z_1 = w_{11}x_1 + w_{12}x_2 + ... + w_{1N}x_N = \sum_{p=1}^{N} w_{1p}x_p$$

donde x_p son las variables explicativas, y Z_1 la variable a explicar. Los coeficientes w_{1p} son:

$$w_{1p} = \frac{cov(x_p, y)}{\sum_{p=1}^{N} cov^2(x_p, y)}$$

El modelo *PLS*, tiene bastante relación con la reducción de componentes principales (*PCA*) que se explicará más adelante, en el hecho de que también aplica una reducción de dimensionalidad antes de aplicar el regresor lineal. La diferencia principal es que PLS es aprendizaje supervisado.

3.1.3. Medición el ajuste del modelo

El ajuste de un modelo se refiere a la capacidad que tiene un modelo de *machine learning* de generalizar datos parecidos a los dados para entrenar. Cuando un modelo está bien ajustado quiere decir que la salida que otorga el mismo, dadas entradas nunca vistas, se aproxima mucho a la salida real o esperada. De igual manera, si un modelo está mal ajustado, su salida no se asemejará a la real. Por lo tanto es necesario que para que un modelo esté bien ajustado, tiene que ser capaz de generalizar lo suficiente con los datos dados y no ajustarse demasiado ni a ellos ni al ruido (posible causa de *overfitting*), pero tampoco ser demasiado simple, puesto que a la hora de introducir datos complejos puede que no los clasifique bien. En la imagen 3.1 se pueden ver los diferentes límites de decisión en función de como esté el modelo ajustado.

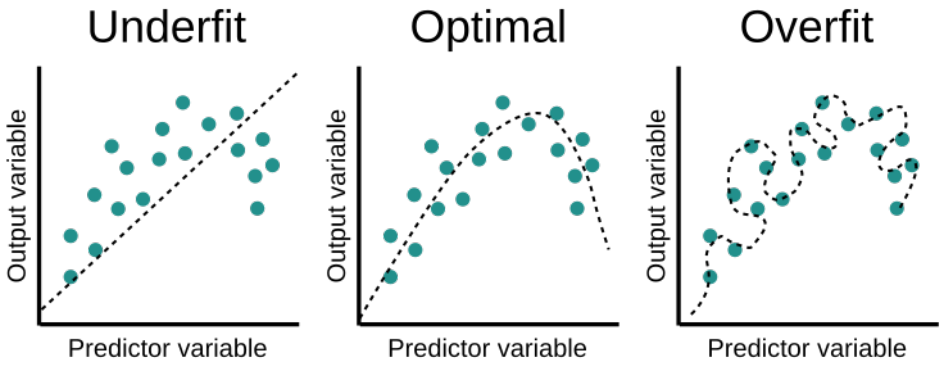

Figura 3.1: Underfitting, óptimo y overfitting (Imagen extraída de fastaireference [4])

Una de las formas más típicas de medir si un modelo está bien ajustado es calculando la raíz del error cuadrático medio (**Root Square Error**) que se define como $RSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$. Para saber cómo de bueno es este método lo compararemos con el método más simple, que consiste en ir comparando cada elemento con la media, ecuación que coincide con la varianza: $\frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y})^2$. Para compararlos únicamente los dividiremos, y obtendremos así el conocido como coeficiente de determinación R^2.

$$R^2 = 1 - \frac{RSE^2}{Var(y)} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}$$

Este coeficiente indica la calidad del modelo para replicar los resultados. El coeficiente de determinación variará generalmente sus resultados entre 0 y 1. Así, cuando se aproxima más a 1, mayor resultará el ajuste del modelo a la variable que se pretende aplicar para el caso en concreto. Por el contrario, cuando adquiere resultados que se acercan al valor 0, menor será el ajuste del modelo a la variable que se pretende aplicar y, justo por eso, dicho modelo resultará menos fiable.

3.1.4. Redes neuronales

Las redes neuronales son un modelo computacional el cual se inspira en las conexiones entre neuronas que forman el cerebro de los animales y en las mismas neuronas. Si bien los ordenadores son actualmente máquinas muy potentes, capaces de realizar cientos de millones de cálculos en segundos, el cerebro humano está mejor optimizado para ciertas tareas, como por ejemplo el reconocimiento de rostros o lugares. Así pues, el trabajo de las neuronas en el complejo sistema que es el cerebro animal, es principalmente transmitir impulsos nerviosos. Estos impulsos se

envían de neurona en neurona, con diferentes intensidades y patrones, los cuales a día de hoy son muy complicados de entender.

Modelo neuronal de McCulloch-Pitts

En el año 1943, de mano de Warren McCulloch (neurólogo y cibernético) y Walter Pitts (lógico en el campo de la neurociencia computacional), nació el primer modelo, que mediante elementos simples de cómputo, trata de emular el funcionamiento de una de estas neuronas.

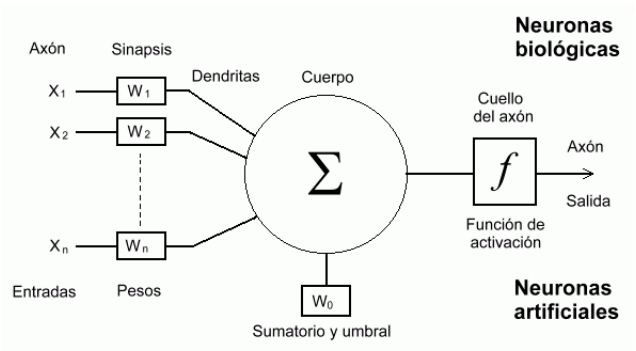

Figura 3.2: Neurona McCulloch-Pitts (Imagen extraída de optica inaoep [5])

Como podemos ver en la imagen 3.2, la neurona está formada por un conjunto de partes simples que se relacionan con las neuronas biológicas:

Valores de entrada: Se trata de un vector de valores, lo cuales representan los valores de entrada de la neurona a través de las dendritas.

Peso de las conexiones: Se trata de un conjunto de valores, asociado a cada conexión entre neuronas, representado como w_{ij} (conexión entre la neurona i y la neurona j). Representan como de fuerte es la conexión entre las neuronas en el cerebro, es decir, la sinapsis.

Función de propagación: Una función principal, la cual reúne la información que ha sido trasmitida desde las *"Dendritas"*, y las aplica una función u operación (una agregación por ejemplo), para que el valor devuelto sea posteriormente tratado. Funciona como el cuerpo de la neurona biológica.

Función de activación : Una función que retorna un valor de salida, variando el valor de entrada. Análogamente al axón de las neuronas biológicas, que no porque les llegue un estímulo han de mandar ellas uno similar.

En general, el modelo que habitualmente se usa es aquel cuya regla de propagación es igual a la suma ponderada de las entradas con sus pesos respectivos. La función de activación f, proporciona su salida y aun así se le añade un parámetro adicional θ_i, conocido como umbral o *bias*, al cual se le pueden dar diferentes usos dependiendo del modelo. El modelo se puede representar por lo tanto de la siguiente manera, para cada entrada x:

$$f(y_j) = f(\sum_{i=1}^{N} w_{ij}x_i + \theta_i)$$

Siendo N el número de variables del modelo. Por lo general el valor del bías se toma como si fuera un peso w_0 con un $x_0 = -1$.

El verdadero poder de los modelos de redes neuronales radica en que a pesar de que las neuronas como elemento base tengan baja capacidad de procesamiento, realizando conexiones variadas entre esta, y diferentes arquitecturas se puede resolver casi cualquier problema.

Perceptrón multicapa (Multi-layer Perceptron)

El perceptrón multicapa (*MLP* por sus siglas en inglés), es un modelo de aprendizaje supervisado el cual aprende una función $f(\cdot) : R^m \rightarrow R^o$ a base de entrenar en un *dataset*, donde m es el número de dimensiones de entrada (el tamaño del vector de datos que se usan como *input*), y o el número de parámetros de salida.

La principal ventaja de este algoritmo, es que la red neuronal está formada por múltiples capas, de manera que tiene capacidad de, dadas unas entradas x y una salida y, ser capaz de aprender funciones de aproximación no lineales, tanto para clasificación como para regresión. Estas capas, que se encuentran entre la primera (*capa de entrada*) y última (*capa de salida*) son denominadas *capas ocultas*. Cada una de estas capas modifica los valores que le llegan de la capa anterior mediante transformaciones lineales y una función de activación no lineal $g(\cdot) : R \rightarrow R$. La capa de salida, recibe los valores de la última capa oculta y los transforma en un valor de salida válido.

Un perceptrón multicapa con una única capa oculta se puede representar con la siguiente ecuación:

$$y_j = \sum_{k=1}^{q} w'_{jk} z_k + \theta'_j = \sum_{k=1}^{q} w'_{jk} f(\sum_{i=1}^{t} w_{ki} x_i + \theta_k) + \theta'_j$$

Donde:

- f es la función de activación.
- x_i son las entradas a la red.
- y_j son las salidas de la red.
- z_k son las salidas de la capa oculta.
- w_{ik} son los pesos para cada neurona de la capa oculta.
- θ_k son los umbrales de los pesos w_{ik}.
- w'_{jk} son los pesos para cada neurona de la capa de salida.
- θ'_j son los umbrales de los pesos w'_{jk}.

El aprendizaje de un perceptrón multicapa se realiza minimizando una función de error, que mide la diferencia entre la salida obtenida y la salida deseada. A menor sea esta diferencia mejor será nuestra red. El proceso de entrenamiento de nuestra red se puede realizar de dos maneras, las cuales básicamente representarán cada cuánto hemos de ajustar los pesos de nuestra red.

1. **Batch** u **offline**. El modelo se entrena con todos los datos antes de dar una predicción y cambiar los pesos. Si los datos cambian, o se añaden nuevos, hay que entrenar de nuevo todo el modelo.

2. **On-Line** o después de cada ejemplo. El modelo se entrena con cada uno de los nuevos datos que se reciben, o en pequeños lotes llamados *mini-batch*. Son muy adecuados para cuando todos los datos no se pueden cargar en la memoria, o para ir haciendo cálculos con datos que se obtienen en tiempo real.

Regularización de MLP

Uno de los principales problemas cuando se desea modelar un modelo neuronal es que no se puede saber de manera previa que estructura (cantidad de capas ocultas y neuronas en cada una de ellas) será la que mejor se adapte al problema que se plantee. Por lo general, la forma más óptima es por experiencias previas del diseñador, por lo que no hay un método claro.

De manera más práctica, se utiliza el parámetro alpha, α, también conocido como término de penalización, que combate el *overfitting* restringiendo el tamaño de los pesos. Aumentar alpha puede corregir una alta varianza (típicamente se ve cuando hay *overfitting*) fomentando pesos más pequeños. Del mismo modo, la disminución de alpha puede corregir un sesgo elevado (un signo de *underfitting*) fomentando pesos más grandes, lo que puede dar lugar a un límite de decisión más complicado. Diferentes ejemplos de esto se pueden ver en la imagen 3.3.

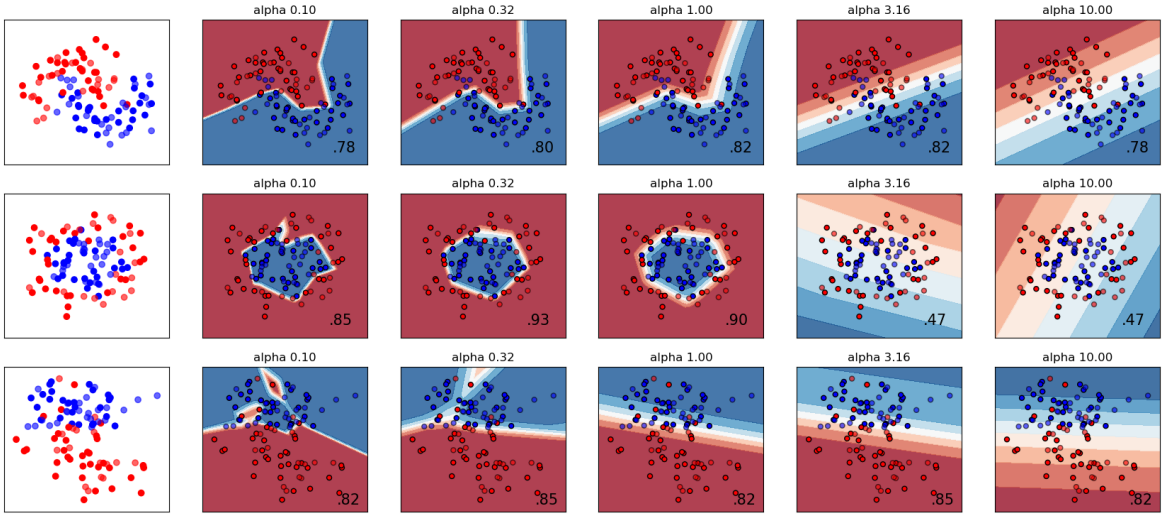

Figura 3.3: Efecto de las variaciones en alpha (Imagen extraída de scikit-learn [6])

Trabajo práctico con MLP

Para la realización de este proyecto, se ha utilizado la librería de **Scikit-Learn** con varias de las herramientas para análisis de datos y modelos de predicción que ofrece la misma. En el caso de las definiciones de MLP se ha usado el objeto *MLPRegressor*. A continuación se explicarán los diferentes cambios que se pueden realizar sobre los hiperparámetros más importantes del objeto para adecuarlo más al problema.

- **hidden_layer_sizes**: Con este parámetro creamos la arquitectura de la red neuronal, especificando tanto el número de capas ocultas de esta, como de neuronas que tiene cada una.
- **activation**: Define la función de activación que van a tener las neuronas de las capas ocultas; ofrece cuatro posibilidades diferentes.
  - *identity* $\Rightarrow f(x) = x$
  - *logistic* $\Rightarrow f(x) = 1/(1 + exp(-x))$
  - *tanh* $\Rightarrow f(x) = tanh(x)$

  - *relu* $\Rightarrow f(x) = max(0, x)$
- **solver**: Diferentes formas de obtener los pesos óptimos.
  - lbfgs: Método de optimización quasi-Newtoniano que solamente necesita el gradiente y la función de pérdida.
  - sgd: Simple, pero muy eficiente para ajustar clasificadores y regresores lineales bajo funciones de pérdida convexas.
  - adam: Probablemente el que tiene mejores resultados de manera general. Converge de manera rápida usando *momentum* y *adaptive learning*.
- **alpha**: El parámetro de regularización explicado en la sección anterior.
- **batch_size**: El tamaño de los lotes, para realizar el entrenamiento según se ha explicado en la sección anterior.
- **learning_rate** y **learning_rate_init**: Controlan cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo, y la tasa de aprendizaje inicial respectivamente. Ofrece tres opciones.
  - *constant*: Aprende de manera constante basándose en lo establecido en learning_rate_init.
  - *invscaling*: Disminuye gradualmente la tasa de aprendizaje a medida que avanza el tiempo.
  - *adaptive*: Mantiene la tasa de aprendizaje constante mientras la pérdida de entrenamiento siga disminuyendo. Cada vez que dos épocas consecutivas no consiguen disminuir la pérdida de entrenamiento en al menos *tol*, o no consiguen aumentar la puntuación de validación en al menos *tol* si *early_stopping* está activado, la tasa de aprendizaje actual se divide entre cinco.
- **max_iter**: Número máximo de iteraciones que hará el algoritmo. El algoritmo se detendrá cuando llegue a este número o cuando converja (en un valor determinado por *tol*)
- **shuffle**: Indica si mezclar o no los elementos en cada iteración, para dar más variabilidad.
- **random_state**: Dado que la generación de números aleatorios parte de una semilla, con este hiperparámetro se puede marcar qué semilla usar.
- **tol**: Se define como la tolerancia para la optimización. Cuando el error no mejora en al menos tol en *n_iter_no_change* iteraciones el algoritmo se detiene.
- Valores de regularización del solver *adam*, como son **beta_1** y **beta_2** que representan la tasa de decrecimiento exponencial para las estimaciones del vector de primer y segundo momento respectivamente. Deben estar en [0, 1). **Epsilon** es un valor para regular la estabilidad numérica.
- **n_iter_no_change**: Máximo número de épocas que se realizarán hasta mejorar en función del parámetro *tol*.

En la sección de experimentos se ahondará más en el trabajo que se ha hecho con este objeto.

3.2. Reducción lineal de dimensionalidad por componentes principales

En el aprendizaje automático uno de los problemas más comunes es el del uso de conjuntos de datos excesivamente grandes, complicando así el manejo y la velocidad a la que se tratan los datos. Para evitar esto existe el concepto de *reducción de dimensionalidad*, que consiste en transformar los datos de una gran dimensión espacial a una menor, manteniendo las propiedades más importantes de los datos iniciales. Idealmente se tratará de llegar a la denominada dimensión intrínseca, que se define como el número de variables mínimo necesario para representar los datos.

Como se ha explicado con anterioridad PCA (Principal Component Analysis), es uno de los métodos estadísticos que nos permite simplificar la complejidad de un conjunto de datos. Supongamos que existe una muestra X, de d individuos los cuales tienen N variables. PCA nos permite encontrar un número Z de variables, siendo siempre $Z < N$ que dan una información lo más cercana posible a las N variables iniciales (A más componentes, más cercano). A cada uno de estos Z valores se le denomina componente principal. Es importante mencionar que se trata de un método no supervisado, es decir, no se usa ninguna información sobre la variable a predecir.

Los usos más frecuentes de PCA son:

- **Reducir la dimensionalidad** para evitar columnas redundantes. Con seleccionar solo unas pocas componentes principales se puede explicar la mayor parte de la variación del *dataset*.
- **Crear nuevas variables no observables**, puesto que hay variables que no se pueden medir directamente, pero si relacionar con otras mediciones. Estas variables no medibles se denominan *indicadores sintéticos*.
- **Análisis exploratorio**, para descubrir relaciones entre los datos.

Gracias a PCA podemos usar una nueva técnica para tratar nuestros datos llamada *Regresión de las componentes principales* (PCR por sus siglas en inglés), la cual se divide en dos pasos:

1. Aplicar PCA a los datos, para tratar reducir la dimensionalidad del *dataset*.
2. Entrenar una regresión lineal sobre los datos transformados.

Cálculo de las componentes principales

Cada una de las componentes principales (Z_i) se obtendrá en este caso, mediante la combinación lineal de las variables originales. Es importante normalizar las variables al inicio, para que así las variables con más varianza no dominen al resto. Como resultado de esto, este algoritmo puede no funcionar tan bien como otros en *datasets* donde el *target* esté altamente relacionado con variables con una varianza pequeña.

La primera de las componentes principales será la que más información contenga, puesto que contendrá la mayor variabilidad de los datos. Se calcula de la siguiente manera:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{N1}X_N$$

los valores $\phi_{11}, ..., \phi_{N1}$ se denominan *loadings*, y son los que dán valor a la componente. Por lo tanto ϕ_{N1} es el *loading* de la variable X_1 en la N componente principal.

Dado un conjunto de datos X, para calcular la primera componente principal se seguirán los siguientes pasos:

- **Centrar las variables:** se le resta a cada valor el valor de la media de la variable, así todas tendrán media 0.
- **Encontrar el valor de los *loadings***: mediante un problema de optimización, maximizando la varianza.

La segunda componente principal (Z_2), será una combinación lineal de las variables, que recoja la segunda dirección con más varianza de los datos, pero evitando que esté correlacionada con Z_1. Gráficamente esto se puede entender como que la dirección cumpla que $Z_2 \perp Z_1$. Es decir la segunda componente principal recogerá toda la varianza que no ha podido recoger la primera variable. Este proceso se repetirá de forma iterativa hasta calcular todas las posibles componentes principales, o hasta que se decida detener el proceso, porque se recoge un porcentaje de variabilidad suficiente.

Matriz de covarianza

Para entender el análisis de componentes principales más en profundidad, debemos mencionar la matriz de covarianza. Se trata de una matriz simétrica $N \times N$, donde N es el número de variables, y se almacena la varianza o covarianza de las mismas. En el caso de tener solo dos variables, se definiría así:

$$\sum = \begin{bmatrix} Var(x) & Cov(x,y) \\ Cov(y,x) & Var(y) \end{bmatrix}$$

Dado que $Cov(x,y) = Cov(y,x)$, la matriz es simétrica, con las varianzas de las diferentes características. Si la covarianza es mayor que cero, las variables están correlacionadas de manera positiva, que implica que se desplazan en la misma dirección, y de manera negativa si es inferior a cero. Si la covarianza es igual a 0, entonces, las variables son independientes.

La matriz de covarianza por lo tanto define tanto la propagación (varianza), como la dirección (covarianza). Por lo tanto con esta matriz podemos representar otros elementos, como por ejemplo un vector, que indique la dirección en la que van los datos y un número que indique el módulo de este. Estos elementos se denominan respectivamente *eigenvector* y *eigenvalue*. En el método PCA, cada una de las componentes se corresponde con un *eigenvector*, y el orden de las componentes se establece por orden decreciente de *eigenvalue*. Así pues, la primera componente es el *eigenvector* con el *eigenvalue* asociado más alto.

Eigenvectores y eigenvalores

Los *eigenvectors* son un caso particular de multiplicación entre una matriz y un vector,

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = 4 \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

donde el vector resultante de la multiplicación, es un múltiplo real del vector original. Los *eigenvectors* de una matriz son todos aquellos vectores que, al multiplicarlos por dicha matriz, resultan en un múltiplo real del mismo.

Cuando se multiplica una matriz por alguno de sus *eigenvectors* se obtiene un múltiplo del vector original. Al valor por el que se multiplica el *eigenvector* resultante se le conoce como *eigenvalue*.

PCA como diagonalización de la covarianza

PCA se puede definir también como diagonalizar la matriz de covarianza, lo que implica que tenemos que encontrar una combinación lineal no trivial de nuestras variables originales de forma que la matriz de covarianza sea diagonal (matriz cuadrada con todos los valores nulos menos

la diagonal principal). Una vez hecho esto, las variables resultantes no están correlacionadas, es decir, son independientes. Se realizarán los siguientes pasos para calcular las combinaciones lineales mencionadas:

1. Situar los datos como una matriz $m \times n$, a la que llamaremos X, donde n es el numero de medidas que se toman sobre los datos (columnas en un *dataset*), y m el número de muestras (filas del *dataset*).
2. Restar la media a cada muestra para obtener la matriz $X' = X - \overline{X}$.
3. Calcular los *eigenvector* de la matriz de covarianza de X'

Con los *eigenvectores* (que corresponderán a una componente principal) calculados obtendremos los *eigenvalues* de la matriz. Si estos *eigenvalues* son grandes, entonces la componente es importante para describir los datos. Se puede ver una representación gráfica de como se representa la matriz de covarianza, los *eigenvectors* y los *eigenvalues* en la imagen 3.4.

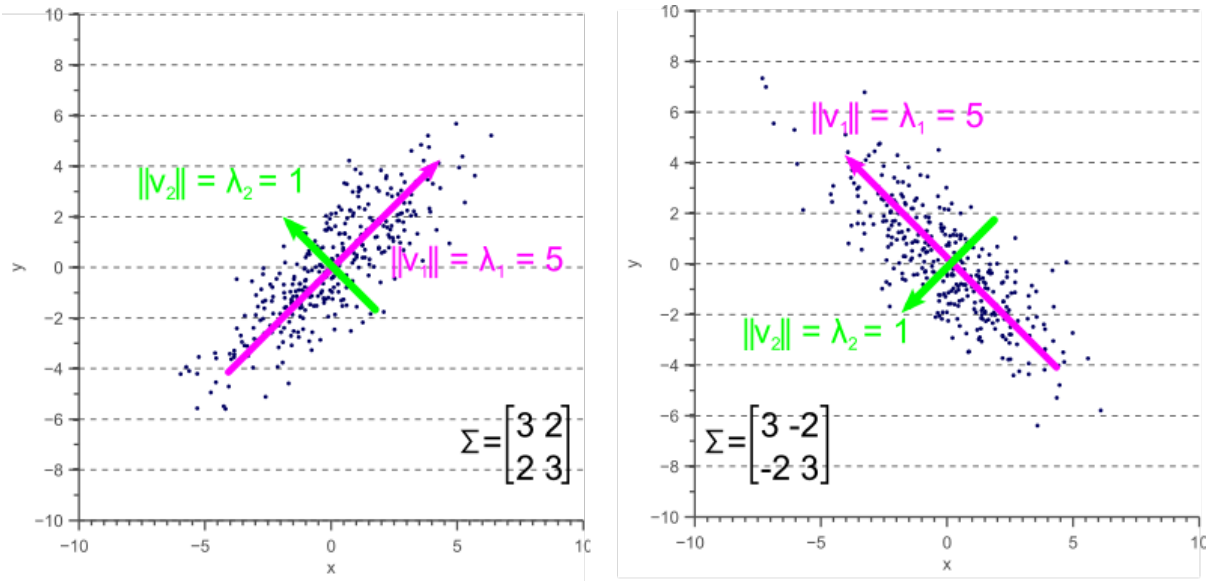

Figura 3.4: Interpretación geométrica de la matriz de covarianza, eigenvectors y eigenvalues (Imagen extraída de visiondummy [7])

4

Experimentos

El objetivo principal de este trabajo es el de aplicar los métodos explicados en el capitulo anterior para la resolución de problemas relacionados con la energía eólica. Nos centraremos principalmente en la predicción del viento en el parque eólico de Sotavento, al Noroeste de la península Ibérica, situado entre los ayuntamientos de Xermade (Lugo) y Monfero (A Coruña).

4.1. Entorno de ejecución

Software

Todo el trabajo realizado en este proyecto a nivel de programación, será realizado usando el lenguaje *Python*. Gracias al entorno de *miniconda*, instalaremos la versión de *python 3* en nuestro entrono, así como las diferentes librerías que usaremos en el código, en concreto:

NumPy: Da soporte para crear vectores y grandes matrices, junto con una gran cantidad de funciones matemáticas de alto nivel para operar con ellas.

Scikit Learn: Es una biblioteca de aprendizaje automático que soporta el aprendizaje supervisado y no supervisado. También proporciona varias herramientas para el ajuste de modelos, el preprocesamiento de datos, la selección y evaluación de modelos y muchas otras utilidades.

Joblib: Librería que proporciona diferentes herramientas de *pipelining*, aunque nosotros principalmente lo usaremos para el volcado de objetos en ficheros, y lectura de estos.

Re: Este módulo proporciona operaciones de coincidencia de expresiones regulares similares a las encontradas en *Perl*.

Además de estas librerías, se ha utilizado tal y como se explica en la sección de formatos de datos meteorológicos, la librería **Pandas**, la cual es una herramienta de análisis y manipulación de datos rápida, potente, flexible y fácil de usar, con la que podremos crear tablas ordenadas por fecha para su posterior tratamiento. Todo el código creado para este proyecto se encuentra en un repositorio de *github*.

Hardware

Para la ejecución de la mayor parte de los scripts necesitaremos una gran capacidad de cómputo, puesto que se tratarán de programas complejos con varias partes las cuales a su vez tendrán un gran coste computacional. Como realizar esto en equipos personales, posiblemente conllevaría unos tiempos de ejecución excesivamente altos, se habilitó una cuenta de estudiantes en el equipo *login1.ccc.uam.es* al cual será posible la conexión mediante *ssh*.

Los códigos se han ejecutado en dos máquinas diferentes, que pertenecen al Centro de Computación Científica (CCC), y se nos permite el acceso por el interés que tiene este trabajo para el propio centro. La primera se trata de una máquina compartida destinada a trabajos con una gran carga, la forma de ejecución entre usuarios se organiza a través de *slurm*, y utilizando los comandos *sbatch, scancel y squeue* para reservar espacio en la cola de ejecución, cancelar procesos y revisar la cola respectivamente. La segunda es una máquina local del CCC llamada *casarrubelos*, de la cual podemos obtener las especificaciones técnicas de la máquina ejecutando comandos *bash* como *lscpu o free*:

Procesador: 56 procesadores - 14 cores por procesador.

Memoria: 755 GB Memoria RAM - 19 GB Memoria SWAP

Sistema Operativo: CentOS Linux 7 (Core)

4.2. Implementación

En esta sección se tratarán los diferentes objetos que nos ofrecen las librerías de *Python 3* que se han mencionado antes y cómo se han usado en específico en este proyecto.

Estandarización de las variables

La estandarización, tipificación o normalización de las variables es un proceso muy típico que nos ayuda a comparar datos que estén en diferentes medidas o que sean de diferentes distribuciones de datos. Este cálculo se realiza de manera muy simple, restando a la variable que interesa la media de la distribución y dividiendo el resultado entre la desviación típica.

$$\hat{X}_i = \frac{X_i - \overline{X_i}}{\sigma(X_i)}$$

La estandarización de los *dataset* es un requisito común para muchos estimadores de aprendizaje automático implementados en **scikit-learn**, puesto que podrían comportarse mal si las características individuales no se parecen más o menos a los datos estándar distribuidos normalmente. Para realizar este proceso la misma librería nos ofrece la clase *StandardScaler*. Esta clase implementa un *Transformer* para calcular la media y la desviación estándar en un conjunto de entrenamiento, y luego poder re-aplicar la misma transformación en el conjunto de prueba. Esta clase es, por tanto, adecuada para su uso en los primeros pasos de un *Pipeline*.

Pipeline

El propósito del *pipeline* es reunir varios pasos para que puedan ser aplicados de forma secuencial mientras se establecen diferentes parámetros. Así pues, el objeto *Pipeline* de **scikit-learn**, aplica secuencialmente una lista de transformaciones sobre los datos y finalmente un

estimador. Los métodos que transforman los datos, han de implementar los métodos *fit* y *transform*, y el estimador final, únicamente el método *fit*. Usar *Pipeline* es muy útil, ya que suele haber una secuencia fija de pasos en el procesamiento de los datos; en nuestro caso estos pasos consistirán en el estandarizar los datos, aplicar luego PCA, y por último el estimador. Por lo tanto el *Pipeline* tiene varias utilidades.

- **Comodidad y encapsulación:** Solo tiene que llamar a *fit y predict* una vez sobre sus datos para ajustar toda una secuencia de estimadores.
- **Selección conjunta de parámetros:** Puede buscar en la cuadrícula los parámetros de todos los estimadores de la cadena de producción a la vez.
- **Seguridad:** En *cross validation* aseguran que se utilizan las mismas muestras para entrenar los transformadores y los predictores.

Transformed TargetRegressor

En los problemas de regresión en los que se debe predecir un valor numérico, también puede ser crítico escalar y realizar otras transformaciones de datos en la variable objetivo. Esto se puede lograr en Python utilizando la clase *TransformedTargetRegressor* de la librería **scikit-learn**. Esta clase transformará la variable objetivo utilizando los mismos datos del *training dataset* usados para ajustar el modelo, y luego aplicará esa transformada a la inversa en cualquier nuevo dato proporcionado al llamar a *predict*, devolviendo las predicciones en la escala correcta. Para utilizar el *TransformedTargetRegressor*, se define especificando el modelo y el objeto de transformación a utilizar en el objetivo, tal y como se puede ver en la figura 4.1.

```
1 y_transformer = StandardScaler()
2 inner_estimator = TransformedTargetRegressor(regressor=regr_base,
3                                               transformer=y_transformer)
4
```

Figura 4.1: Ejemplo de definición del Transformer para el target

Grid Search y KFolds

Grid-searching es el proceso de exploración de los datos para encontrar los parámetros óptimos de un modelo determinado. Este proceso no es únicamente para un tipo de modelo, sino que es muy versátil, a pesar de conllevar un alto coste computacional. *Grid-Search* construirá un modelo para cada combinación de parámetros posible, lo recorrerá, y generará unos resultados.

Como en todo el proyecto haremos uso de la librería de **scikit-learn** con el objeto *GridSearchCV*. A este le pasaremos un diccionario con los parámetros que querremos ir variando en el modelo (un ejemplo de esto se puede ver en la figura 4.2). Como se ha mencionado antes, *GridSearchCV* prueba todas las posibilidades, por lo que el coste computacional aumentará de manera exponencial, a medida que añadamos parámetros que combinar entre ellos en el modelo.

Para validar el modelo, *GridSearchCV* hace uso de *cross-validation*, donde se separan los datos en *train* y *test*. Para realizar estas separaciones utilizamos el objeto *KFolds*, el cual dado un conjunto de entrenamiento lo divide en k conjuntos más pequeños, y para cada uno de los k conjuntos realizan las siguientes dos operaciones:

```
param_grid = {
    'regressor__mlp_m__alpha':  [10.**k for k in range(-5, 5)],
    'regressor__pca__n_components': list(range(50, 501, 50)),
}
```

Figura 4.2: Ejemplo de posible entrada de parámetros a explorar

- Se entrena el modelo usando $k-1$ de los conjuntos como *training*.
- El modelo resultante se valida con el conjunto sobrante.

El rendimiento final es la media de los valores obtenidos en la validación para cada uno de los k conjuntos.

4.3. Predicción en sobre el parque eólico de Sotavento

El Parque Eólico Experimental Sotavento, situado entre los ayuntamientos de Xermade (Lugo) y Monfero (A Coruña), se ha convertido en un referente en la investigación y divulgación de las energías renovables, del ahorro energético y de su interrelación con el medio ambiente. Sotavento realiza actividades principalmente en las áreas de producción de energías renovables, investigación y formación. [13]

Utilizaremos los datos que nos proporciona la cátedra UAM-IIC sobre el campo eólico de Sotavento, para probar el modelo propuesto de reducción de dimensionalidad en un conjunto de datos pequeño en comparación con el objetivo final del trabajo. Este conjunto de datos tiene un tamaño de 8760 muestras, una por cada hora del año (24×365) y 3480 variables medidas en diferentes puntos. Con esto, se quiere decir, que realmente se miden 5 variables diferentes, las cuales ahora se comentarán, pero las mismas se miden en diferentes puntos geográficos, siguiendo el modelo de rejilla que se ha explicado en capítulos anteriores. La distancia a la cual se van tomando medidas se toma variando la latitud y la longitud en 0.125^{o}, que equivale a 7' y 30" (en el sistema de grados, minutos y segundos) y a unos 13,850 kilómetros. Las medidas que se tienen en cuenta son las siguientes:

- Dirección del viento desde el Oeste a una distancia de 10 metros del punto geográfico (10u).
- Dirección del viento desde el Sur a una distancia de 10 metros del punto geográfico (10v).
- Modulo de la velocidad del viento a 10 metros del punto geográfico (vel10).
- Dirección del viento desde el Oeste a una distancia de 100 metros del punto geográfico (100u).
- Dirección del viento desde el Sur a una distancia de 100 metros del punto geográfico (100v).
- Módulo de la velocidad del viento a 100 metros del punto geográfico (vel100).
- Presión en la superficie (sp).
- Temperatura a 2 metros (2t).

Adicionalmente se tiene la fecha y hora en la que se ha tomado cada medición, realizándose una cada hora durante todo un año (prediction date), en nuestro caso lo utilizaremos como índice.

Estos puntos están centrados en el campo eólico, siendo las esquinas noreste y suroeste los puntos (44^{o},-9.5^{o}) y (42.25^{o},-6^{o}) respectivamente. El valor a predecir será la potencia que produce el campo eólico, medida en megavatios. Para escalar este valor en un rango de 0 a 1, dividiremos todos los datos por la potencia máxima que alcanza el parque (17.56 MW). [14]

Los datos de meteorología ofrecidos por la cátedra UAM-IIC del parque eólico de Sotavento, comprenden los años 2016, 2017 y 2018; por lo tanto la estrategia que seguiremos para la validación de nuestros modelos será la de entrenarlos con 2016 y 2017, los cuales actuarán como conjuntos de entrenamiento y validación respectivamente, y tratar de predecir 2018 que será nuestro conjunto de test. Por lo tanto, teniendo dos años para entrenar, el dataset, tendrá un tamaño de 17544 filas y 3481 columnas.

4.3.1. Pasos previos

Como se ha comentado anteriormente, se realizarán diferentes *scritps* para probar las diferentes partes de las que se formará un programa final. Todos estos *scritps* serán escritos en Python.

Paso 1: Conversión a dataframe

En este problema, debido a que tenemos los datos en el formato CSV, la transformación a *DataFrame* será directa, empleando la función *read_csv* ajustada para que la primera columna del fichero CSV, que se corresponde con el momento en el que se realizó la medición, funcione como índice y en formato *datetime* de **pandas**. Este mismo proceso se realizará con el conjunto de datos de *target*, dejando ambos ordenados por fecha y hora.

Paso 2: Creación del Pipeline

Tal y como se ha mencionado con anterioridad, el uso de *Pipelines* es muy una práctica muy frecuente en la implementación de soluciones de *machine learning*, debido a lo común que es normalizar los datos antes de su uso para la predicción, así como otras transformaciones. En nuestro caso, crearemos un objeto *Pipeline*, el cual nos permite definir los pasos, así como un nombre para estos mediante el parámetro *step* del mismo, el cual consiste en una lista de tuplas nombre-transformación, las cuales han de implementar los métodos *fit/transform*.

En nuestro caso comenzaremos normalizando los datos con *StandarScaler*, y tras esto usando *PCA* haremos una reducción de dimensionalidad por componentes principales, para finalmente aplicar el regresor que consideremos. Fuera del *Pipeline* trasformaremos también la variable objetivo con *TransformedTargetRegressor* con las mismas transformaciones que se han realizado para el *training dataset*.

Paso 3: Hiperparametrización

En este paso es donde está la mayor carga computacional del programa. Usando el objeto *GridSearchCV* crearemos un espacio de búsqueda en el cual ir probando con diferentes parámetros, y usando *KFolds* estableceremos el número de subconjuntos en los que vamos a probar. En este problema haremos 2 *folds*, lo que quiere decir que entrenaremos con, aproximadamente, un año y validaremos con otro. Es importante recalcar que evitaremos (mediante el parámetro *shuffle=False* de *KFolds*) que los conjuntos de datos se mezclen antes de realizar los subconjuntos, asegurándonos así de que aún en diferentes ejecuciones ya sea con diferentes o el mismo modelo, los datos usados para entrenar y validar son los mismos.

Por otro lado, gracias a *GridSearchCV* crearemos un espacio de búsqueda con diferente cantidad de hiperparámetros dependiendo del modelo que queramos investigar. En el caso de *PCA*, que será el transformador común de todos los modelos que probemos dado que el objetivo

principal de este trabajo es ver qué tal se comporta la reducción de dimensionalidad con este problema, hiperparametrizaremos el número de componentes principales con las que transformaremos el *dataset*. Dado que tratar de ir componente a componente viendo el valor más óptimo es computacionalmente demasiado costoso, intentaremos reducir la cantidad de casos a contemplar. Dado que sabemos que PCA selecciona variables en función de la varianza que dan al conjunto de datos, computaremos PCA una vez, y comprobaremos la suma acumulada de la varianza, para ver cuando empieza a llegar a una saturación. El punto en el cual esto ocurra, será en el que debemos movernos para encontrar un número de componentes principales que sea capaz de explicar bien el *dataset*, pero sin escoger demasiadas para no tener un conjunto de datos muy grande.

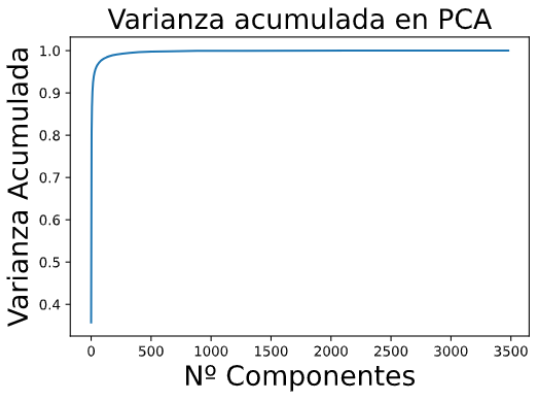

Figura 4.3: Suma acumulada todas las componentes.

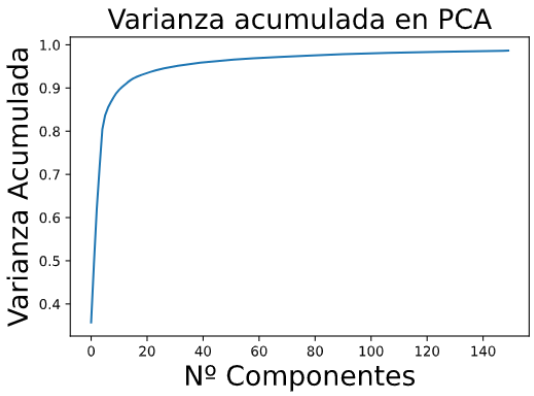

Figura 4.4: Suma acumulada primeras 150 componentes

Como se puede apreciar en la figura 4.4 (el cual es un *zoom* de la figura 4.3), en las primeras 80 componentes ya se obtiene una varianza bastante cercana a 1, por lo que nos moveremos en este rango.

Para valorar lo bien que funciona el modelo planteado, utilizaremos como función de puntuación el cálculo del error absoluto medio (estableciendo en el objeto *scoring* = *neg_mean_absolute_error*), el cual se calcula de manera similar al error cuadrático medio, que se ha explicado en capítulos anteriores.

$$\hat{e}(f) = \hat{e}_s(f) = \frac{1}{N}\sum_{p=1}^{N} |y^p - f(x^p)|$$

En las gráficas que se muestran a continuación se toma el valor positivo del MAE para mayor claridad.

Paso 4: Reducción del tamaño de puntos medidos

Inicialmente, nos interesa realizar pruebas con los datos, para asegurarnos de que las herramientas usadas se ajustan a las necesidades del problema, mientras que el resultado de las predicciones no nos interesa tanto. Por esto mismo, realizaremos una reducción de dimensionalidad manual, en la cual eliminaremos algunas columnas del *dataframe*, haciendo que el área en la que tomamos mediciones se reduzca considerablemente.

Como se ha explicado antes, las medidas están centradas en el parque eólico de Sotavento, por lo que para mantener el punto central estable, estableceremos un punto central (con valor 43.375 longitud, -7.875 latitud), y a partir de este estableceremos un área nueva, con 121 *grid*

points, de los cuales sacaremos 8 mediciones, haciendo un total de 968 columnas de nuestro nuevo *DataFrame* (El inicial tiene 435 *grid points*). Gracias a este cambio, sin cambiar la base del problema, reducimos bastante el coste computacional que tendrán las siguientes operaciones.

Para hacer esta reducción de dimensión, utilizaremos el paquete de operaciones con expresiones regulares de python (**re**). Gracias a que las columnas del *dataset* están organizadas por el valor que están midiendo y el punto de la coordenada, crearemos una lista de los nuevos puntos que nos interesan a nosotros, y con una expresión regular, eliminaremos todos los que no estén en esa lista de nuestro *DataFrame*.

Una vez todo sea funcional, y comprobemos que el código creado no tiene fallos y cumple con su función, nos saltaremos este paso para trabajar con el conjunto de datos completo. No mostraremos los datos obtenidos al realizar regresiones con este *dataset* reducido, puesto que únicamente se utilizará para probar que todo funcione.

4.3.2. Modelos aplicados al problema

Modelo Ridge

Para nuestra primera aproximación al problema utilizaremos como estimador final del *Pipeline*, será el modelo *Ridge*. El modelo *Ridge* nos permite crear una regresión lineal para relacionar los datos de entrada con la salida. Usaremos este modelo de manera inicial para ver si el trabajo realizado en código funciona tal y como queremos, puesto que debido a su simplicidad no aumentará mucho la carga computacional de la solución implementada. También nos servirá para obtener resultados simples con los cuales comparar a medida que implementemos nuevos modelos, tanto resultados numéricos como temporales.

Estableceremos el modelo Ridge como la base para comparar resultados, así que inicialmente realizaremos una regresión normal, sin realizar reducción de dimensión, para ver que resultados obtenemos, y así poder comparar con los datos obtenidos luego aplicando *PCA*. Una vez aplicado este modelo vemos que el MAE en validación nos da 0.0676, y prediciendo el el año 2018 obtenemos un error del 0.06809. En la figura 4.5 se puede ver una dispersión de puntos, que representan la calidad de la predicción, en la que la linea azul representa una predicción perfecta.

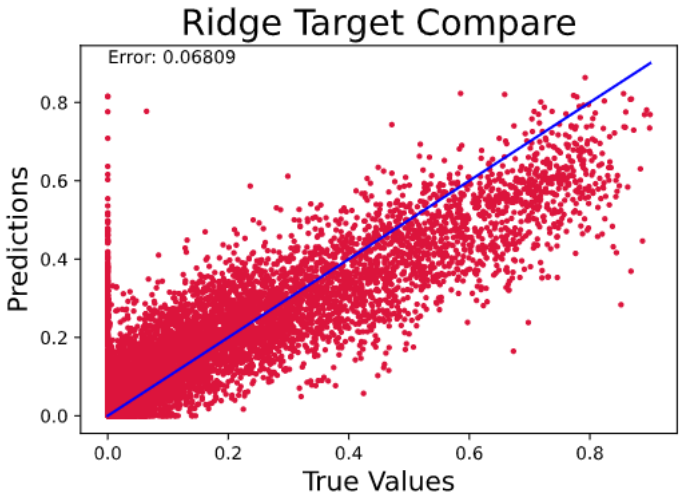

Figura 4.5: Comparación del target predicho para 2018 y el real usando Ridge

En este modelo, una vez aplicamos PCA, tendremos que hiperparametrizar dos parámetros: el *número de componentes* en el caso de la transformación de PCA, y el valor regulador *alpha*

el modelo. Recordemos que al variar el parámetro *alpha* y hacerlo más grande, la regularización será mayor, puesto que se penalizará más la varianza de las variables. Al hacer una búsqueda de cuál es el valor óptimo de estos dos parámetros, obtenemos que de los valores probados, los que mejor resultado dan son siempre cercanos al máximo de componentes probadas y un *alpha* muy elevado. Podemos ver estos valores en las figuras 4.6 y 4.7. Es importante recalcar que el tiempo en el que se han obtenido estos datos (utilizando 5 cores para la ejecución en una máquina de las características antes descritas) ha sido de aproximadamente 30 minutos.

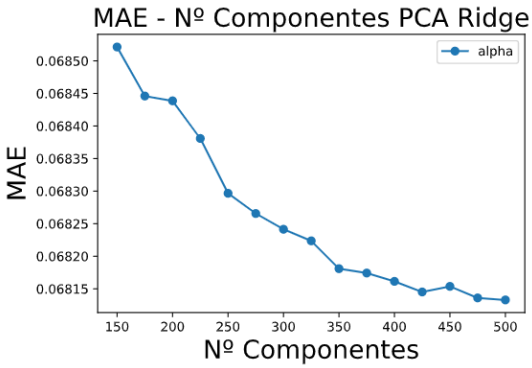

Figura 4.6: MAE para cada valor probado de componentes principales con el alpha óptimo

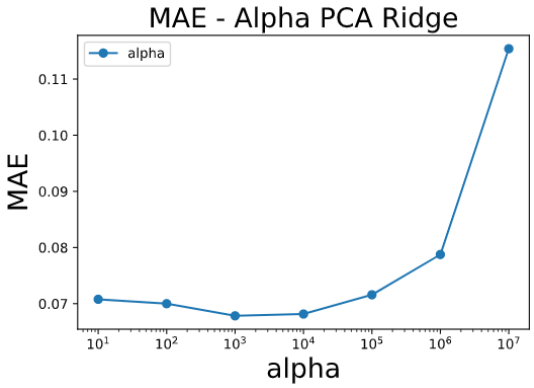

Figura 4.7: MAE para cada valor probado de alpha con las componentes principales óptimas

Como podemos ver, con los parámetros óptimos obtenemos un MAE en validación de 0.0678. Aun así este modelo no es muy fiable, puesto que no ajusta mucho la variable predicha con la real. Al realizar el *test* con el año 2018, obtenemos un MAE de 0.06917 (podemos ver una nube de puntos viendo la calidad de la predicción en la figura 4.8).

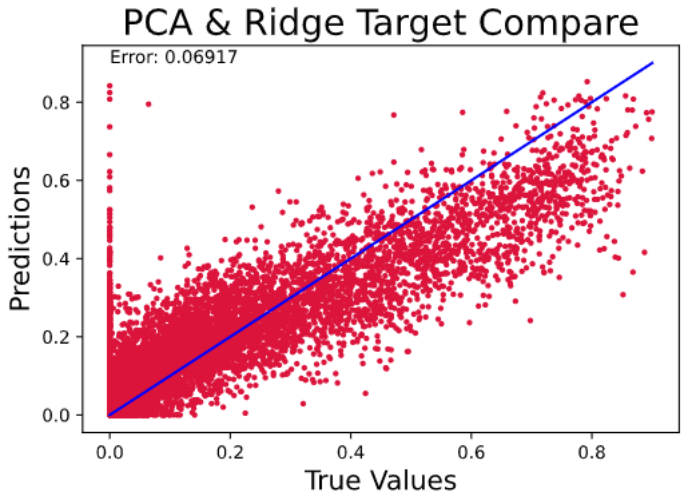

Figura 4.8: Comparación del target predicho para 2018 y el real usando PCA y Ridge

Modelo Partial Least Squares

Usaremos el modelo *Partial Least Squares*, para comparar de manera directa con los valores obtenidos en *Ridge*, puesto que tienen una funcionalidad similar. Sin embargo en este modelo únicamente hiperparametrizaremos el número de componentes principales con las que queremos hacer la reducción de dimensionalidad del conjunto de datos, y no podremos suponer que el número óptimo de componentes será el mismo que para *PCA*. Esto es así porque *PLS* funciona

de manera diferente en el cálculo de las componentes principales (ver subsección 3.1.2), y se trata además de aprendizaje supervisado, por lo que va comparando directamente con los resultados reales.

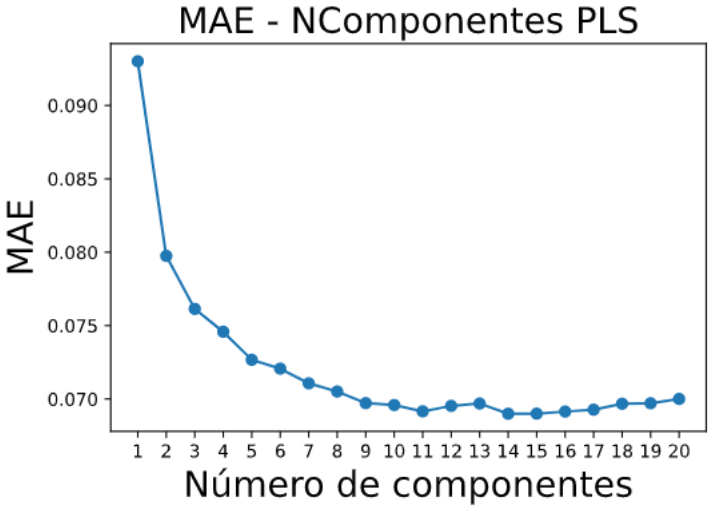

Figura 4.9: Score obtenido para diferentes números de componentes en PLS

Como se puede ver figura 4.9, el número óptimo de componentes principales para este problema son 14, y se obtiene un MAE de 0.06899 en validación, que como se preveía es muy similar al obtenido utilizando el modelo Ridge. Aun así *PLS*, tarda considerablemente menos, debido a que trabaja con muchísimas menos componentes, y sólo tenemos que parametrizar un valor. Aun así podemos comprobar en la figura 4.10 que tiene un rendimiento similar a *Ridge* con *PCA*, con un MAE en test de 0.06870.

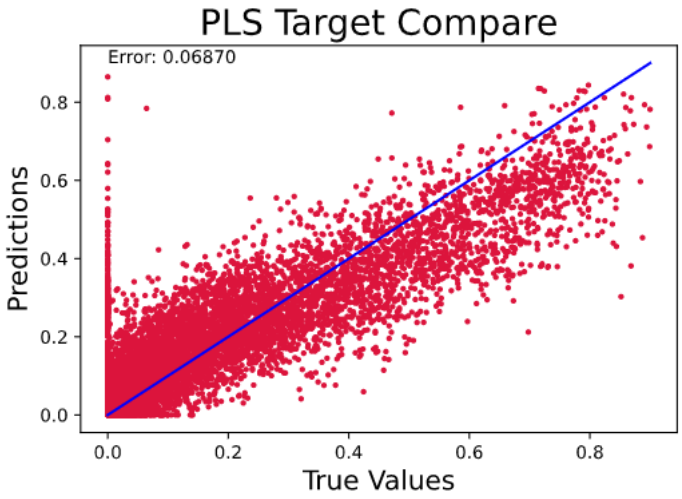

Figura 4.10: Comparación del target predicho para 2018 y el real usando PLS

Modelo Multi Layer Perceptron

Finalmente utilizaremos el modelo más complejo explicado, *MLP*. Para este modelo, al igual que con *Ridge*, primero ejecutaremos sin reducción de componentes principales (sin *PCA*), y a continuación tendremos que hiperparametrizar tanto el número de componentes como un valor *alpha* que servirá como término de penalización. Según la documentación de **scikit-learn** [15],

se penaliza usando el algoritmo L2, que al igual que en *Ridge* penaliza en base al cuadrado de los coeficientes. El resultado del test sin reducción de componentes es de 0.0693 (en la figura 4.11 podemos ver una nube de puntos representando la calidad de predicción del modelo *MLP*).

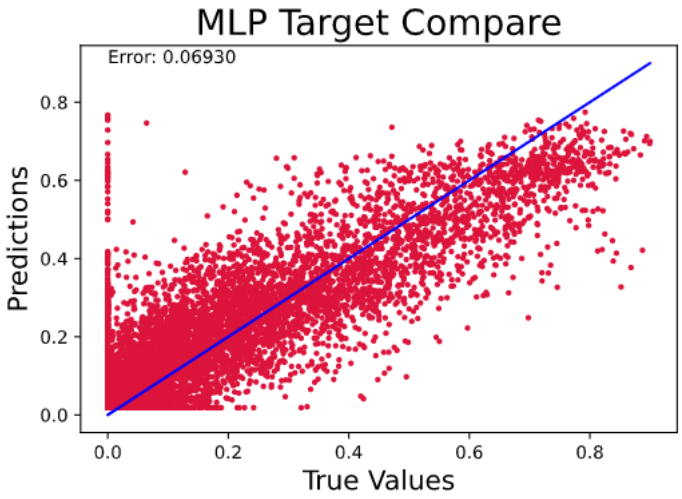

Figura 4.11: Comparación del target predicho para 2018 y el real usando MLP

En este caso los parámetros óptimos serían 375 componentes principales, así como un parámetro de regularización *alpha* igual a 10. Podemos ver estos valores en las figuras 4.12 y 4.13 respectivamente. Con estos valores en validación obtenemos un MAE de 0.05764.

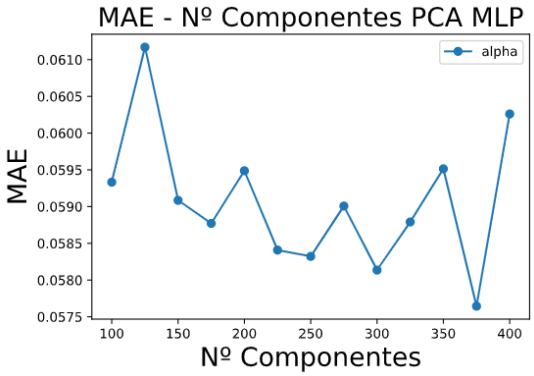

Figura 4.12: MAE para cada valor probado de componentes principales con el alpha óptimo

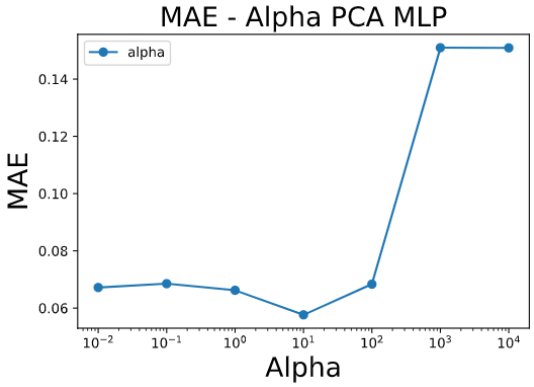

Figura 4.13: MAE para cada valor probado de alpha con las componentes principales óptimas

Con los perceptrones multi capa, hemos también de plantear una arquitectura de la red, es decir, el número de capas ocultas que tendrá la red, así como el número de neuronas que tendrá cada una de estas capas. Como se ha explicado en capítulos anteriores, esta decisión muchas de las veces no se decide en base a ningún valor a excepción de la experiencia del diseñador y la complejidad que queramos darle a cada capa. En nuestro caso tras probar con varias configuraciones encontramos un buen resultado en una red neuronal con dos capas ocultas, teniendo estas 200 y 100 neuronas respectivamente. Entrenando y validando con los años 2016 y 2017, y prediciendo el 2018, al igual que con los modelos anteriores, obtenemos un MAE de validación de 0.05764 y de test de 0.6478 (en la figura 4.14 podemos ver una nube de puntos representando la calidad de predicción del modelo *MLP* con reducción de dimensionalidad).

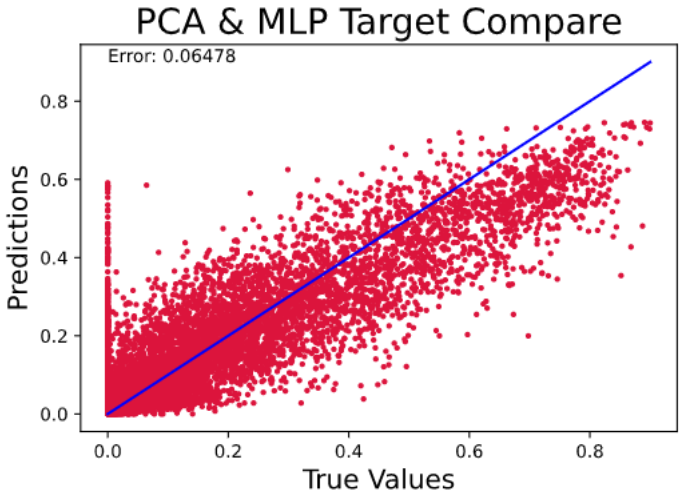

Figura 4.14: Comparación del target predicho para 2018 y el real usando PCA y MLP

4.4. Discusión sobre los resultados del parque eólico de Sotavento

Todos los datos relevantes obtenidos en los experimentos previos se muestran en la tabla 4.1 de manera conjunta. Observando los mismos somos capaces de ver que, comparando con el modelo simple de *Ridge* sin aplicar reducción de dimensionalidad, los resultados suelen ser de peor calidad si usamos un modelo poco flexible y reducción de dimensionalidad. Al reducir la dimensionalidad de un conjunto de datos, a pesar de optimizar el proceso con métodos numéricos, como con *PCA*, siempre se perderá un poco de información, y por lo tanto la precisión de las predicciones puede bajar. En contraposición, al reducir la dimensionalidad, la velocidad a la que se ejecutaran muchos de los algoritmos de *machine learning* sobre esos datos reducidos, será mayor. En el caso de la predicción meteorológica, que recolecta tantos datos, y necesita predicciones rápidas, puede que sea más rentable aumentar la velocidad de predicción y bajar la precisión. Aun con esto, hay casos donde la información que se pierde no es relevante para la resolución del problema, y modelos que sean más flexibles son capaces de mejorar la predicción con respecto su caso base, llegando así a la situación óptima, donde se reduce la dimensión, resolviendo así problemas de rendimiento, y eliminando complejidad del problema, haciendo que algunos modelos mejoren la predicción.

La diferencia entre el rendimiento de los diferentes estimadores se puede apreciar en la tabla 4.1. Con modelos como *Ridge*, que es muy simple, si se realiza reducción de dimensionalidad siempre se cogerán las componentes máximas posibles, para evitar cualquier pérdida de información, y aún con esto no se llega a la misma calidad de predicción que sin reducir componentes. Por lo tanto podemos deducir, que si usamos modelos simples y pocos flexibles, unicamente podremos reducir el tamaño de los datos para aumentar el rendimiento, a coste de reducir la calidad de predicción. En otro caso, usando un modelo más flexible y complejo, como *MLP Regressor*, la reducción de componentes resulta óptima, puesto que se reduce el tamaño, aumentando la velocidad a la que se aplican los algoritmos, y además al eliminar datos que pueden ser redundantes, y el algoritmo ser capaz de adaptarse a los datos restantes y ser lo suficientemente flexible, se aumenta la calidad de la predicción.

	Número de componentes	Alpha	MAE Validación	MAE Test 2018
Ridge	-	1000	0.0676	0.06809
PCA & Ridge	275	10000	0.0678	0.06917
PLS	11	-	0.06899	0.06870
MLP	-	10	0.06153	0.06930
PCA & MLP	175	10	0.05764	0.06478

Cuadro 4.1: Valores obtenidos para cada regresor aplicado al problema de Sotavento

4.5. Aplicación en la Peninsula Ibérica

En esta última sección aplicaremos todo lo visto anteriormente a la datos tomados en toda la península ibérica, donde la cantidad de datos es realmente grande. De manera similar al procedimiento realizado con el campo eólico de Sotavento, el IIC nos proporciona los datos de la península correspondientes a los años 2016, 2017 y 2018. Estos datos también se toman mediante puntos cardinales y de igual manera, separándo las medidas unos 13,850 kilómetros entre ellas. El tamaño de estos *datasets* es de 8760 filas (una por cada hora del año, como se ha explicado antes) y 46783 filas cada uno, siendo bastante más grandes que los usados en la sección anterior.

Para trabajar con estos conjuntos de datos, no se puede usar un equipo o métodos convencionales, puesto que cargar más de 2 de ellos en memoria consume más de 8 *GigaBytes*. Para solucionar esto hemos hecho una aproximación, realizando una reducción de variables de la misma manera que se ha explicado en el **paso 4** de la subsección 4.3.1, pero en este caso, haciendo saltos de 0.5 para todo el conjunto de datos.

De la misma manera que con los datos del parque eólico de sotavento, utilizaremos diferentes modelos para ver los resultados obtenidos. La explicación de los difernetes algoritmos se ha explicado en el apartado anterior de manera más exaustiva, por lo que en este nos centraremos principalmente en ver los resultados obtenidos usando las mismas técnicas que se han usado en el parque eólico de sotavento. Es importante mencionar que las predicciones en este apartado se han realizado sobre la potencia real medida, sin escalar en un rango de [0-1] como en la sección anterior, por lo tanto los ejes se encuentan en megavatios. En la tabla 4.2 si se muestran los valores dentro de este rango, habiendo dividido los valores entre la potencia máxima instalada de la penínisula (23000 MW) [16], para que sea más cómoda la comparación con la tabla 4.1.

	Número de componentes	Alpha	MAE Validación	MAE Test 2018
Ridge	-	1000	0.05640	0.05019
PCA & Ridge	300	1000	0.05626	0.05619
PLS	17	-	0.05613	0.05442
MLP	-	1	0.03399	0.03566
PCA & MLP	125	10	0.03174	0.03297

Cuadro 4.2: Valores obtenidos para cada regresor aplicado al problema de Sotavento

En las figuras 4.15 y 4.16 podemos ver como varía el *mean absolute error* con respecto el número de componentes y el *alpha* respectivamente aplicando *PCA* y *Ridge Regression*. Al igual que en la sección anterior, vemos como se tiende a obtener mejores resultados a medida que aumentamos las componentes, puesto que este algoritmo, como ya se ha explicado, tiende a necesitar toda la información posible, y no es lo suficientemente flexible. En las figuras 4.17 y 4.18 podemos ver una nube de puntos que nos indica lo bien que predicen el modelo de *Ridge* sin aplicar reducción de dimensionalidad y aplicándola respectivamente.

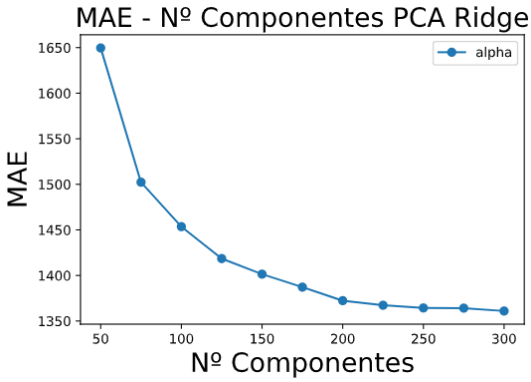

Figura 4.15: MAE para cada valor probado de componentes principales con el alpha óptimo

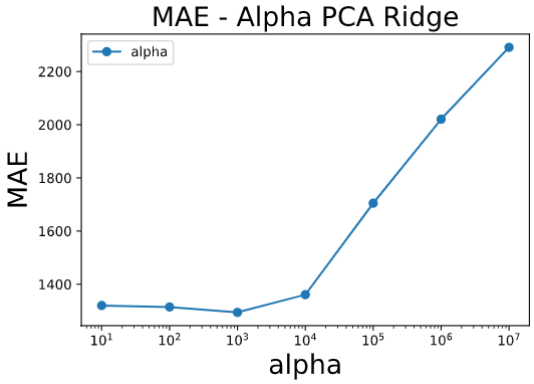

Figura 4.16: MAE para cada valor probado de alpha con las componentes principales óptimas

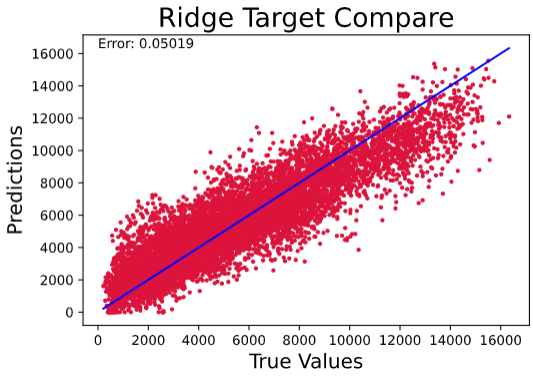

Figura 4.17: Comparación del target predicho para 2018 y el real usando Ridge

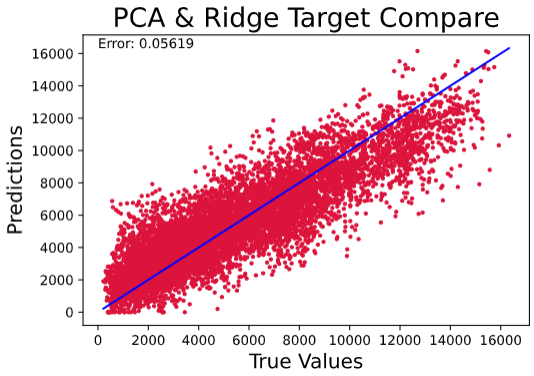

Figura 4.18: Comparación del target predicho para 2018 y el real usando PCA y Ridge

Siguiendo con el orden de la sección anterior, comprobamos el rendimiento de PLS, y comprobamos que el *MAE* varia con respecto el número de componentes de manera similar a lo observado anteriormente, como se puede ver en la figura 4.19 necesitando muchas menos componentes que *Ridge* con reducción de dimensionalidad, para obtener un resultado similar. En la nube de puntos mostrada en la figura 4.20 se puede ver graficamente que la calidad de la predicción no difiere mucho de la obtenida en *Ridge* con reducción de dimensionalidad.

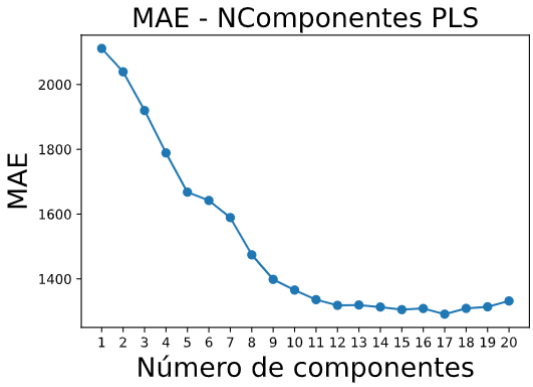

Figura 4.19: Score obtenido para diferentes números de componentes en PLS

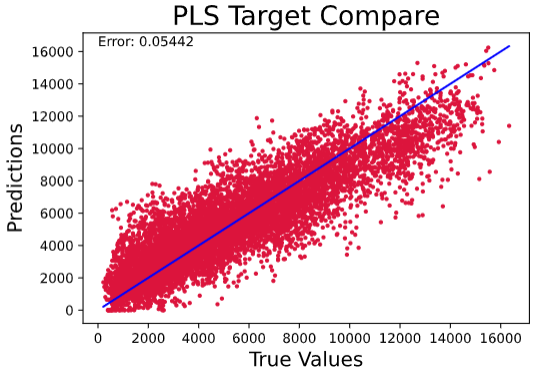

Figura 4.20: Comparación del target predicho para 2018 y el real usando PLS

Finalmente comprobamos el rendimientoque nos da el algoritmo más flexible que hemos probado, *MLP*. De igual manera que en la sección anterior, realizamos la predicción sin realizar reducción de dimensionalidad y posteriormente realizándola. Tal y como podíamos esperar por los resultado obtenidos en los experimentos del parque eólico de Sotavento, realizar una reducción de dimensionalidad con este algoritmo mejora ligeramente la predicción, además de ejecutar más rápido. Al igual que anteriormente, vemos en la figura 4.21 que este algoritmo no necesita obtener siempre el máximo de componentes para tener un gran rendimiento. Finalmente en las figuras 4.23 y 4.24 podemos ver gráficamente mediante una nube de puntos, lo bien que se ajusta este algoritmo a la predicción perfecta, asemejándose bastante más que los algoritmos anteriores.

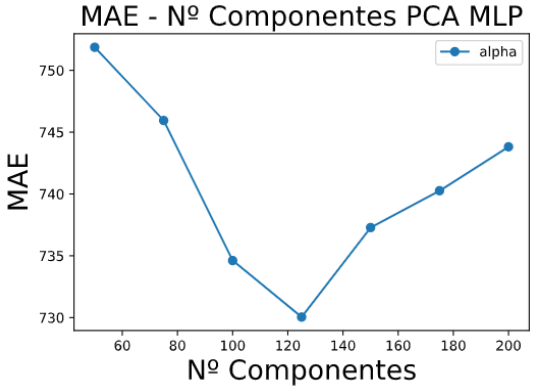

Figura 4.21: MAE para cada valor probado de componentes principales con el alpha óptimo

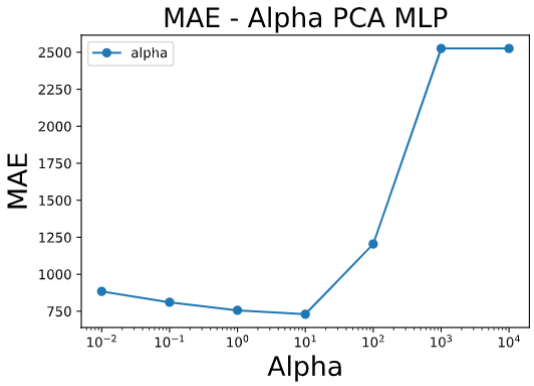

Figura 4.22: MAE para cada valor probado de alpha con las componentes principales óptimas

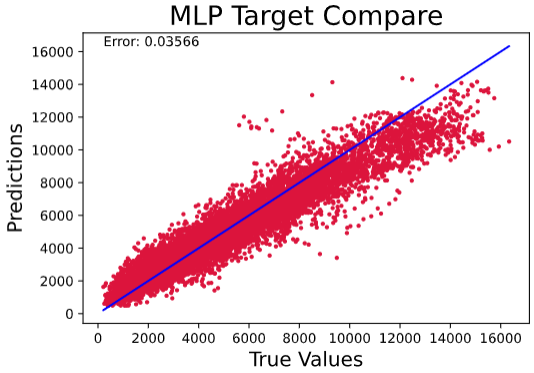

Figura 4.23: Comparación del target predicho para 2018 y el real usando MLP

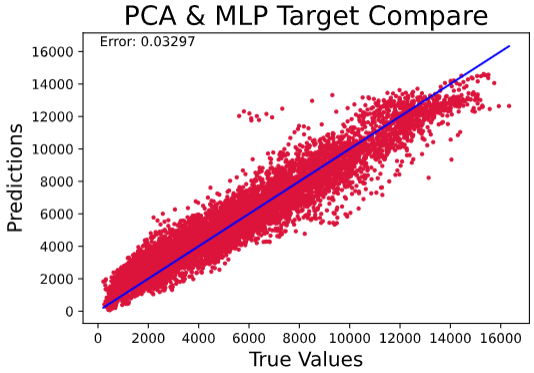

Figura 4.24: Comparación del target predicho para 2018 y el real usando PCA y MLP

5

Conclusiones y trabajo futuro

5.1. Conclusiones

Este trabajo de fin de grado ha servido como una introducción al *machine learning* así como a herramientas teóricas y prácticas para poner los conocimientos a prueba. Se han explicado de manera detallada diferentes acercamientos matemáticos utilizados para resolver problemas de regresión, así como para realizar comparaciones estadísticas entre datos. También se ha adquirido soltura en el uso de librería específicas del lenguaje *Python* muy utilizadas para el manejo de datos y el aprendizaje automático. Debido al problema específico propuesto, se ha explicado también como funciona la predicción meteorológica a través de la predicción numérica del tiempo. Dado que es un problema que tiene una gran cantidad de mediciones, se han explicado también técnicas de reducción de dimensionalidad basadas en los datos numéricos y sus características estadísticas, haciendo bastante inciso en el análisis de componentes principales (PCA).

El objetivo principal era el de comprobar si para el problema de la predicción meteorológica se podían obtener mejores resultados aplicando una reducción de dimensionalidad previa a la predicción, para así reducir el tamaño de los conjuntos de datos, que en este problema son muy elevados, y mejorar las predicciones a la vez. Como se ha podido comprobar esto es posible, aunque no con cualquier modelo, puesto que si el mismo no es lo suficientemente flexible, siempre necesitará la mayor cantidad de información posible, con métodos para reducir la dimensión de los datos, como PCA, siempre se pierde algo. Aun así, con modelos flexibles como los perceptrones multi-capa, sí se puede conseguir optimizar este proceso en algunos casos, como el de este problema.

5.2. Trabajo futuro

Aúnque el trabajo realizado cubre los objetivos fijados al inicio del trabajo, el proceso se puede mejorar, obteniendo así valores aun mejores con los mismos modelos usados en este trabajo, o usar modelos diferentes para comprobar que resultados se obtienen. Una de las acciones que se pueden realizar para optimizar este proceso, es el de separar los tipos de variables que tienen los conjuntos de datos, puesto que estamos reduciendo la dimensionalidad en base a la varianza de datos que no tienen por que tener los mismos tipos de variación, ni se miden de la

misma manera, como por ejemplo la velocidad del viento y la temperatura. Un camino a seguir sería el de separar estos datos, y aplicar *PCA* a cada conjunto por separado.

Otro de los caminos a seguir, podría ser el de realizar los mismos experimentos realizados en la península, pero sin hacer la reducción de los puntos de rejilla que hemos realizado a mano. Esto ha sido principalmente por limitaciones de cómputo, por lo que para solucionar esto se podrían hacer los mismos experimentos en la nube, para ver si los resultados son incluso más favorables.

Bibliografía

[1] Surf forecast. Surface wind on tuesday 11 may at 2pm cest. https://www.surf-forecast.com/maps/Spain/wind/6, 2021.

[2] Frank Singleton. Viewers for grib files. http://weather.mailasail.com/Franks-Weather/Grib-File-Viewers.

[3] OSGeo Live. Netcdf dataset overview. https://live.osgeo.org/archive/10.5/en/overview/netcdf_dataset_overview.html, 2013.

[4] Hugo F.-Montenegro. Overfitting and underfitting. https://www.fastaireference.com/overfitting, 2020.

[5] Héctor García Estrada et al. Sistema de identificación online usando redes neuronales diferenciales. *Actas del Congreso Nacional de Tecnología Aplicada a Ciencias de la Salud*, 1:8, 2018.

[6] scikit-learn developers. Neural network models (supervised). https://scikit-learn.org/stable/modules/neural_networks_supervised.html, 2019.

[7] Vincent Spruyt. A geometric interpretation of the covariance matrix. https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/, 2014.

[8] Asociación Empresarial Eólica (AEE). La energía eólica y sus ventajas. https://aeeolica.org/sobre-la-eolica/la-eolica-y-sus-ventajas.

[9] About Us (European Centre for Medium-Range Weather Forecasts). European centre for medium-range weather forecasts. https://www.ecmwf.int/en/about, 2017.

[10] Anders Persson. User guide to ecmwf forecast products. http://cedadocs.ceda.ac.uk/1218/1/ECMWF_user_guide_2001.pdf.

[11] WMO (World meteorological Organization). A guide to the code form fm 92-ix ext. grib. https://www.wmo.int/pages/prog/www/WDM/Guides/Guide-binary-2.html, 2018.

[12] Página oficial netCDF. Documentación sobre netcdf. https://www.unidata.ucar.edu/software/netcdf/docs/index.html.

[13] Ministerio transición ecológica. Información sobre el parque eólico experimental sotavento. https://www.miteco.gob.es/es/ceneam/recursos/quien-es-quien/parque-eolico-experimental-sotavento.aspx, 2019.

[14] Antonio Hernández Romero. Información sobre el parque eólico experimental de sotavento. http://bibing.us.es/proyectos/abreproy/70692/fichero/9+Parque+Eolico+Experimental+de+Sotavento.pdf, 2017.

[15] scikit-learn developers. Mlp regressor documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html, 2019.

[16] Red eléctrica de España. Sistema eléctrico español. https://www.ree.es/sites/default/files/downloadable/avance_informe_sistema_electrico_2017_v3.pdf, 2017.

[17] scikit-learn developers. Decomposing signals in components (matrix factorization problems). https://scikit-learn.org/stable/modules/decomposition.html#pca, 2019.

[18] Fernando Sancho Caparrini. Redes neuronales: una visión superficial. http://www.cs.us.es/~fsancho/?e=72, 2019.

[19] NCSS Statistical Software. Ridge regression. https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf, 2018.

[20] Valentina Alto. Pca: Eigenvectors and eigenvalues. https://towardsdatascience.com/pca-eigenvectors-and-eigenvalues-1f968bc6777a, 2019.

[21] José Santamarta. Las energías renovables son el futuro. https://www.nacionmulticultural.unam.mx/mezinal/docs/511.pdf.